US Macro Data Forecasting Report

[View on TensorFlow.org] [Run in Google Colab] [View source on GitHub] [Download notebook]

This is a report on analyzing and forecasting the US macro data using $\textbf{Recurrent Neural N}$ $\textbf{Convolutional Neural Network (CNNs)}$ and $\textbf{Generative Adversarial Net}$ report is:

$\textbf{Part I. Statistical analysis}$

- Basic manipulation
- Correlation analysis
- Time series analysis with `ARIMA`

$\textbf{Part II. Deep learning models}$

- Basic model: single-step, single-feature forecasting with LSTM
- Generalized model: multi-step, multi-feature forcasting with LSTM
- Advanced model: Generative Adversarial Network (GAN) with RNN and CNN.

$\textbf{Part III. Conclusions and Next steps}$

- Conclusions
- Next steps

Introduction

1. The Notebook

Follow the notebook, we can recreate all the results, notice that

- Upload the `USMacroData.xls` file to the root folder on google colab.
- To navigate better, use the table of contents bottom on the upper-left sidebar.
- $\textbf{For clarity, all code cells are hiden, double click on the cell to get the}$
- Change the parameters as indicated in the comments to create more custom outputs.
- All source code can also be found in the project file folder

2. The US Macro dataset

This report uses a US Macro Dataset provided by the ADP.

Before analyzing the data with codes, we have the following observations.

- This dataset contains $\textbf{6}$ different features (the $\textbf{Inflation, Wage, Unemployment,}$ $\textbf{InterstRate}$) about the macro economy of the US.

- Data were collected every $\mathbf{1}$ month, beginning in $\mathbf{1965\text{-}01\text{-}01\ to\ 2015\text{-}12\text{-}01}$.
- In total, we have $\mathbf{612\ rows\ (month)}$ and $\mathbf{6\ columns\ (features)}$.

Part I.1 Basic manipulation

Code and examples

```
#@title ```basic.py```

#import numpy, pandas and matplotlib

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')


#Basic checks: find null values and fill, set index, etc.

def basic_check(df, index_name = "Month"):
  """Find the null values and set index of a given DataFrame.
  :param: df, pd.DataFrame, the data, e.g. df = pd.read_excel("USMac
  :param: index_name, str, name of the index, must be one of the col
  :rtype: pd.DataFrame
  """
  df = df.sort_values(index_name)
  df.set_index(index_name, inplace=True)

  #check for null entries
  print("Null values summary:\n")
  print(df.isnull().sum())

  return df

def plot_column(df, feature):
    """Plot the resampled column of df, e.g. plot_column(df, "Inflat

    :param: df, pandas.DataFrame, the data, e.g. df = pd.read_excel(
    :param: feature, str, name of column to be plotted.
    """
    y = df[feature].resample('MS').mean()
    y.plot(figsize=(18, 8))
    plt.xlabel('Date')
    plt.ylabel(feature)
    plt.show()
```

basic.py

```
#@title read the file and show the head

us_macro = pd.read_excel("USMacroData.xls", "All")
us_macro.head()
```

read the file ar

	Month	Inflation	Wage	Unemployment	Consumption	Investment	InterestRa
0	1965-01-01	1.557632	3.200000	4.9	6.972061	12.3	3.(
1	1965-02-01	1.557632	3.600000	5.1	7.811330	13.2	3.(
2	1965-03-01	1.242236	4.000000	4.7	7.828032	18.7	4.(
3	1965-04-01	1.552795	3.585657	4.8	8.477938	9.8	4.(
4	1965-05-01	1.552795	3.968254	4.6	7.139364	10.2	4.'

```
#@title Basic checks: find null values and fill, set index, etc.

us_macro.isnull().sum()
df = basic_check(us_macro)
df.head()
```

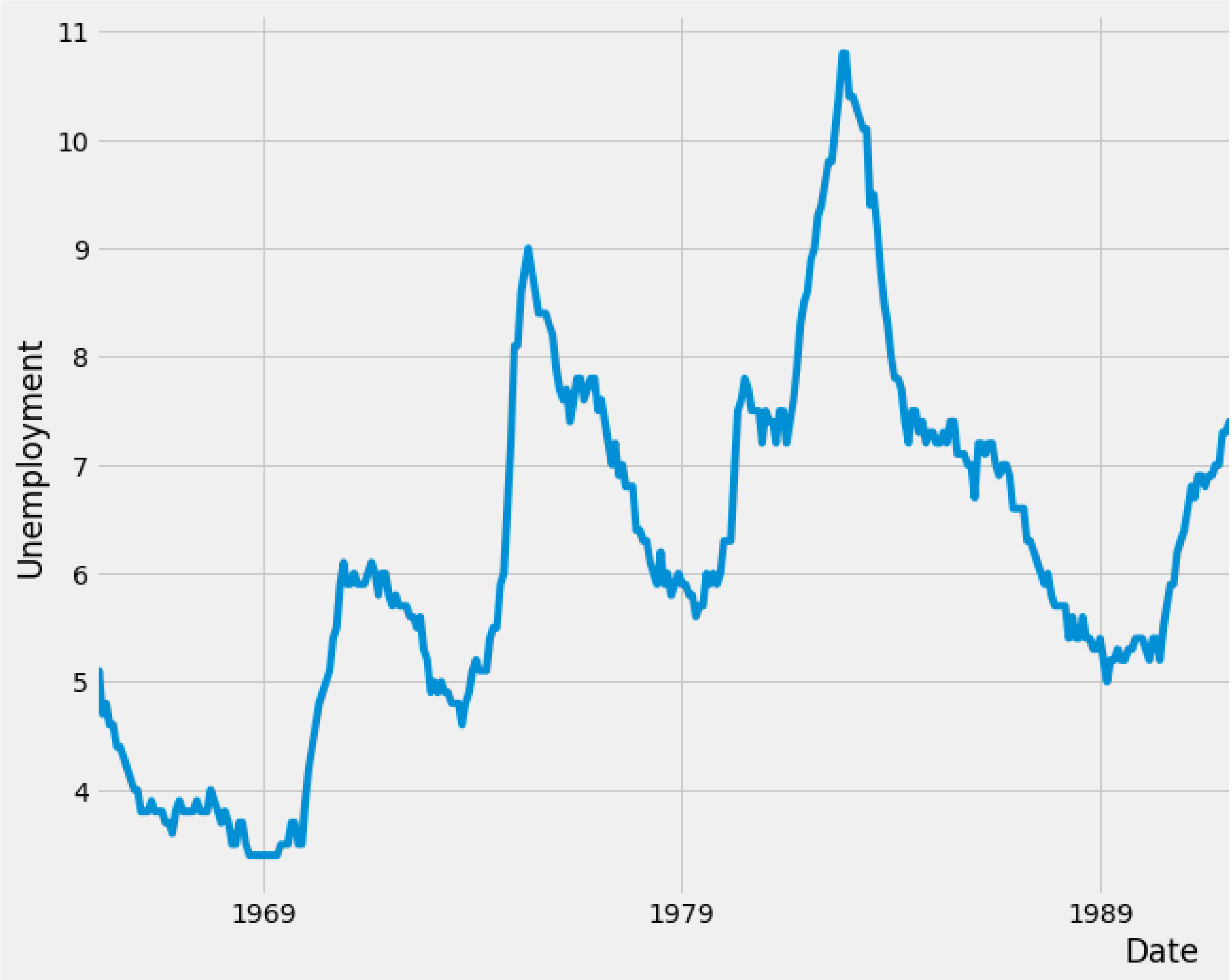
Basic checks:

```
Null values summary:

Inflation       0
Wage            0
Unemployment    0
Consumption     0
Investment      0
InterestRate    0
dtype: int64
```

	Inflation	Wage	Unemployment	Consumption	Investment	InterestRate
Month						
1965-01-01	1.557632	3.200000	4.9	6.972061	12.3	3.90
1965-02-01	1.557632	3.600000	5.1	7.811330	13.2	3.98
1965-03-01	1.242236	4.000000	4.7	7.828032	18.7	4.04
1965-04-01	1.552795	3.585657	4.8	8.477938	9.8	4.09
1965-05-01	1.552795	3.968254	4.6	7.139364	10.2	4.10

```
#@title Example: plot the "Inflation" column
"""
Replace "Inflation" by any feature in our data to get other plot.
"""
plot_column(df, "Unemployment")
```

Example: plot



Data Analysis

As a high level overview, some distinguishable patterns appear when we plot the data:

- **In the 80's (1979-1989), all features experienced some drastic chang**
- The time-series has **seasonality pattern**, for example, **Unemployment** has **long**
  goes through 1 or 2 major up and downs. We will examine the seasonality more carefully in

Part I.2 Correlation analysis

Though it's indicated that there's no obvious correlation among the 6 features, we compute severa
Naive correlation, Pearson correlation, local Pearson correlation, instan
and related statistics in order to

- Test the validity of the assumption (i.e. no two features are apprantly correlated).
- Chose source and target features for later model builds.

By doing so, we can get more understanding about the 'quality' and 'inner relations' of the data. If
explanatory power to the feature that we want to predict (e.g. "Inflation"), then there is no need for
learning models. On the other hand, if one feature has higher-than-random correlations to another

the feature and the other as the target. In this case, **to determine which feature leads,** the **Dynamic time wrapping**.

Code and Examples

```
#@title ```correlation.py```                                          correlation.|

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

#matplotlib.rcParams['axes.labelsize'] = 14
#matplotlib.rcParams['xtick.labelsize'] = 12
#matplotlib.rcParams['ytick.labelsize'] = 12
#matplotlib.rcParams['text.color'] = 'k'

from pylab import rcParams
rcParams['figure.figsize'] = 18, 8


import statsmodels.api as sm
import warnings
import itertools
warnings.filterwarnings("ignore")


import seaborn as sns
import scipy.stats as stats
from scipy.signal import hilbert, butter, filtfilt
from scipy.fftpack import fft,fftfreq,rfft,irfft,ifft

# For the dynamic_time_warping function
!pip install dtw
from dtw import dtw, accelerated_dtw


def pearson(df, feature1, feature2):
    """Compute and plot the overall pearson correlation of feature1 a
    e.g. pearson(df, "Inflation", "Wage") compute and plot the overal

    :param: df, pandas.DataFrame, data contains different features (
    :param: feature1, str, name of the column, e.g. "Inflation"
    :param: feature2, str, name of another column e.g. "Wage"
    """
    overall_pearson_r = df.corr()[feature1][feature2]
    print(f"Pandas computed Pearson r: {overall_pearson_r}")
    # out: Pandas computed Pearson r: 0.2058774513561943

    r, p = stats.pearsonr(df.dropna()[feature1], df.dropna()[feature2
    print(f"Scipy computed Pearson r: {r} and p-value: {p}")
    # out: Scipy computed Pearson r: 0.20587745135619354 and p-value
```

```
    #Compute rolling window synchrony
    f,ax=plt.subplots(figsize=(14,3))
    df[[feature1, feature2]].rolling(window=30,center=True).median()
    ax.set(xlabel='Time',ylabel='Pearson r')
    ax.set(title=f"Overall Pearson r = {np.round(overall_pearson_r,2


def local_pearson(df, feature1, feature2):
    """Compute and plot the local pearson correlation of feature1 an
    e.g. local_pearson(df, "Inflation", "Wage") compute and plot the

    :param: df, pandas.DataFrame, data contains different features (
    :param: feature1, str, name of the column, e.g. "Inflation"
    :param: feature2, str, name of another column e.g. "Wage"

    """
    # Set window size to compute moving window synchrony.
    r_window_size = 120
    # Interpolate missing data.
    df_interpolated = df[[feature1, feature2]].interpolate()
    # Compute rolling window synchrony
    rolling_r = df_interpolated[feature1].rolling(window=r_window_si
    f,ax=plt.subplots(2,1,figsize=(14,6),sharex=True)
    df[[feature1, feature2]].rolling(window=30,center=True).median()
    ax[0].set(xlabel='Frame',ylabel='Smiling Evidence')
    rolling_r.plot(ax=ax[1])
    ax[1].set(xlabel='Frame',ylabel='Pearson r')
    plt.suptitle("Smiling data and rolling window correlation")


def butter_bandpass(lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a



def butter_bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = filtfilt(b, a, data)
    return y

def instant_phase_sync(df, feature1, feature2):
    """Compute and plot the instantaneous phase synchrony of feature
    e.g. instant_phase_sync(df, "Inflation", "Wage") compute and plo

    :param: df, pandas.DataFrame, data contains different features (
    :param: feature1, str, name of the column, e.g. "Inflation"
    :param: feature2, str, name of another column e.g. "Wage"
    """
    lowcut  = .01
    highcut = .5
    fs = 30.
    order = 1
```

```
    d1 = df[feature1].interpolate().values
    d2 = df[feature2].interpolate().values
    y1 = butter_bandpass_filter(d1,lowcut=lowcut,highcut=highcut,fs=
    y2 = butter_bandpass_filter(d2,lowcut=lowcut,highcut=highcut,fs=

    al1 = np.angle(hilbert(y1),deg=False)
    al2 = np.angle(hilbert(y2),deg=False)
    phase_synchrony = 1-np.sin(np.abs(al1-al2)/2)
    N = len(al1)

    # Plot results
    f,ax = plt.subplots(3,1,figsize=(14,7),sharex=True)
    ax[0].plot(y1,color='r',label='y1')
    ax[0].plot(y2,color='b',label='y2')
    ax[0].legend(bbox_to_anchor=(0., 1.02, 1., .102),ncol=2)
    ax[0].set(xlim=[0,N], title='Filtered Timeseries Data')
    ax[1].plot(al1,color='r')
    ax[1].plot(al2,color='b')
    ax[1].set(ylabel='Angle',title='Angle at each Timepoint',xlim=[0
    phase_synchrony = 1-np.sin(np.abs(al1-al2)/2)
    ax[2].plot(phase_synchrony)
    ax[2].set(ylim=[0,1.1],xlim=[0,N],title='Instantaneous Phase Syn
    plt.tight_layout()
    plt.show()



def dynamic_time_warping(df, feature1, feature2):
    """Compute and plot dynamic time warping of feature1 and feature
    e.g. instant_phase_sync(df, "Inflation", "Wage") compute and plo

    :param: df, pandas.DataFrame, data contains different features (
    :param: feature1, str, name of the column, e.g. "Inflation"
    :param: feature2, str, name of another column e.g. "Wage"
    """
    d1 = df[feature1].interpolate().values
    d2 = df[feature2].interpolate().values
    d, cost_matrix, acc_cost_matrix, path = accelerated_dtw(d1,d2, d


    plt.imshow(acc_cost_matrix.T, origin='lower', cmap='gray', inter
    plt.plot(path[0], path[1], 'w')
    plt.xlabel(feature1)
    plt.ylabel(feature2)
    plt.title(f'DTW Minimum Path with minimum distance: {np.round(d,
    plt.show()
```

```
Requirement already satisfied: dtw in /usr/local/lib/python3.6/dist-packages (1.4.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from
```

```
#@title Example: Naive correlation.
```

Example: Naiv

```
df.corr()
```

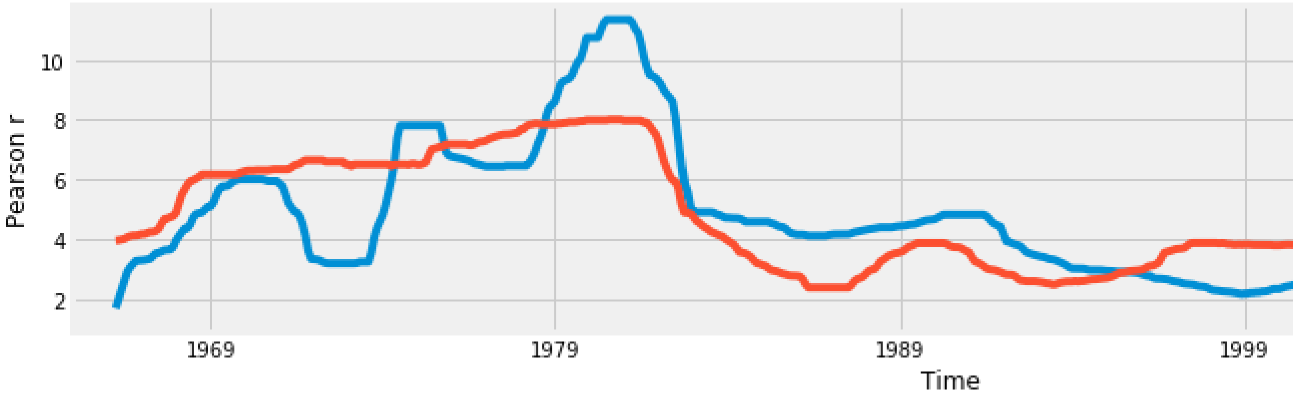
	Inflation	Wage	Unemployment	Consumption	Investment	InterestR
Inflation	1.000000	0.778155	0.191886	0.617820	-0.341421	0.773
Wage	0.778155	1.000000	-0.068529	0.703745	-0.125412	0.647
Unemployment	0.191886	-0.068529	1.000000	-0.097183	-0.038286	-0.027
Consumption	0.617820	0.703745	-0.097183	1.000000	0.203165	0.655
Investment	-0.341421	-0.125412	-0.038286	0.203165	1.000000	-0.234
InterestRate	0.773616	0.647482	-0.027809	0.655305	-0.234573	1.000

```
#@title Example: Pearson correlation
pearson(df, "Inflation", "Wage")
```

Example: Pear

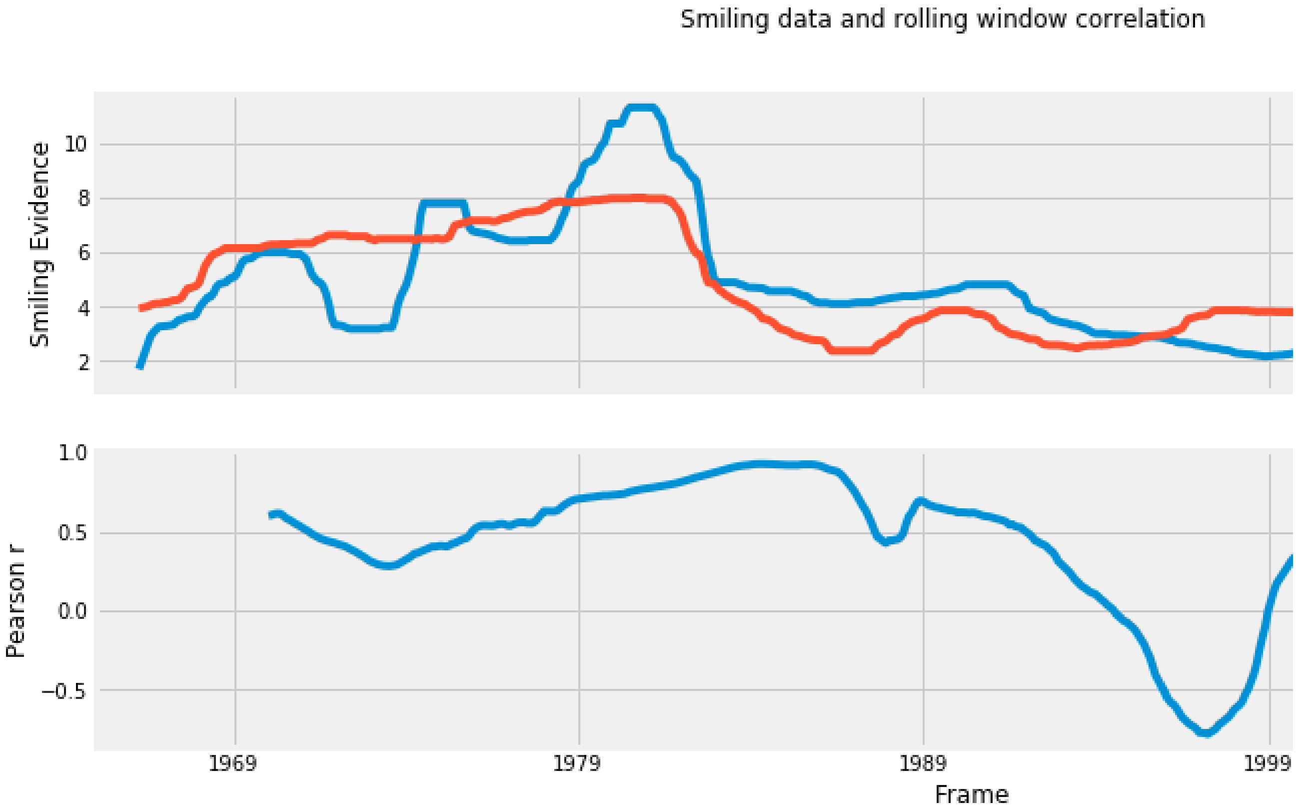
```
Pandas computed Pearson r: 0.7781551675438367
Scipy computed Pearson r: 0.7781551675438365 and p-value: 2.53137614903759e-125
```

Overall Pearson r = 0.78



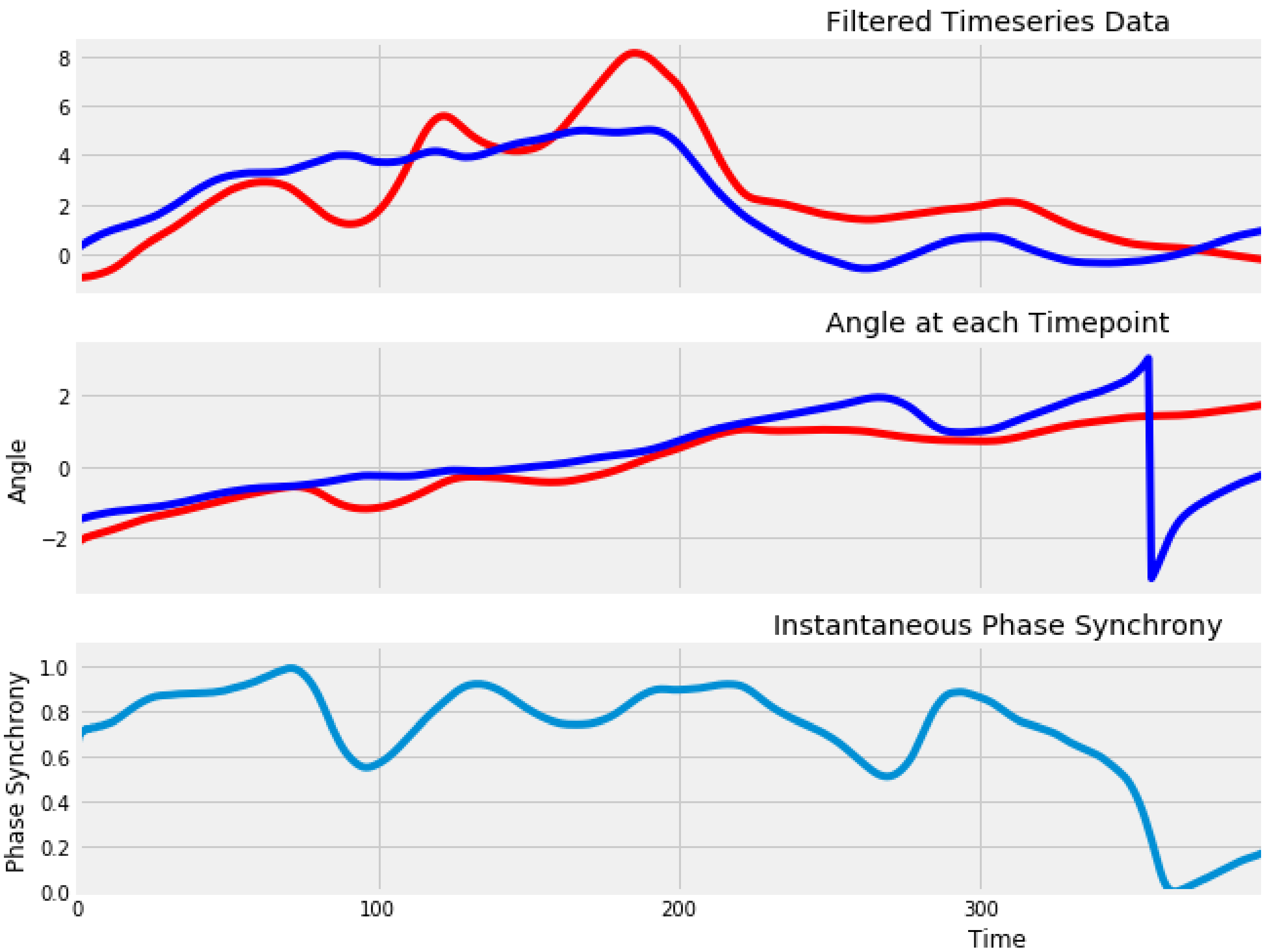

```
#@title Example: local Pearson correlation
local_pearson(df, "Inflation", "Wage")
```

Example: loca



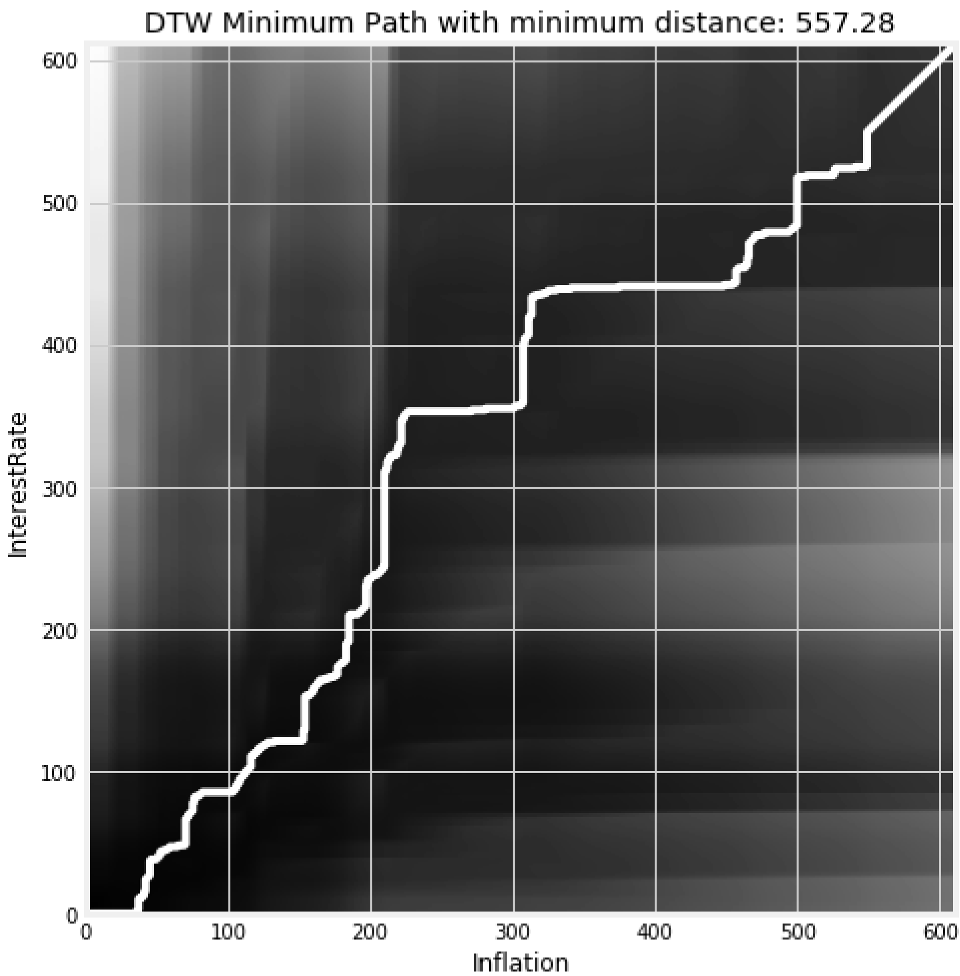

```
#@title Example: instantaneous phase synchronization
instant_phase_sync(df, "Inflation", "Wage")
```

Example: insta



```
#@title Example: dynamic time wraping
dynamic_time_warping(df, "Inflation", "InterestRate")
```

Example: dyna



Data analysis

Inspecting the correlations from different angles, we find

- **Inflation and Wage have the highest correlation, 0.778155,** among all th
- Inflation, Wage, Consumption and IntestRate show quite high positive correlation, and low n Unemployment and Investment.
- **Most features slightly leads the Inflation feature**.
- For the first 30 years, certain feature pairs show **high instantaneous phase synchr**

We conclude that

- **The assumption that no two features have apparent correlation is w**
- It's reasonable to **use Inflation as target and the other 5 features as source for forcastin**
  .

Part I.3 Time series analysis with ARIMA

As we mentioned above, some remarkable patterns (e.g. seasonality pattern) naturally appear in c

- We visualize our data using **time-series decomposition** that allows us to decompos trend, seasonality, and noise.
- We **train an ARIMA (Autoregressive Integrated Moving Average) m** Inflation values. To get optimal output, we first
- Use **grid search** to get the optimal parameters for the ARIMA mode.
- We use **ARIMA diagnostics** to investigate any unusual behavior.

Code and examples

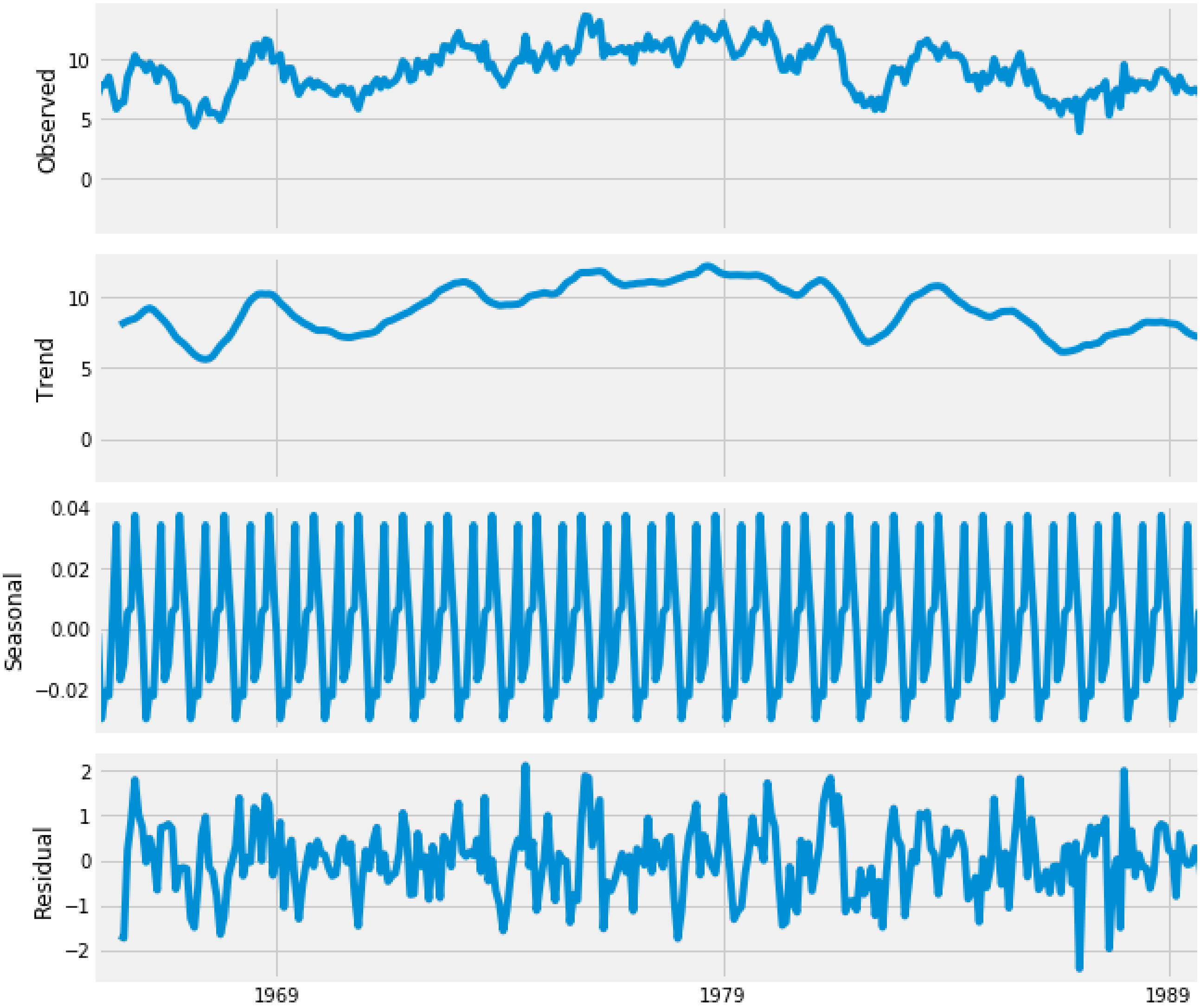
```
#@title ```time_series.py```                                    time_series.|


def plot_column(df, feature):
    """Plot the resampled column of df, e.g. plot_column(df, "Inflat:

    :param: df, pandas.DataFrame, the data, e.g. df = pd.read_excel(
    :param: feature, str, name of column to be plotted.
    """
    y = df[feature].resample('MS').mean()
    y.plot(figsize=(15, 6))
    plt.show()


def plot_component(df, feature):
    """Decompose the time series data into trend, seasonal, and resi

    :param: df, pd.DataFram.
    :param: feature, str,column name/feature name we want to decompo:
    :rtype: None
    """
    decomposition = sm.tsa.seasonal_decompose(df[feature].resample("I
    fig = decomposition.plot()
    plt.show()





###### This section uses ARIMA to analyze the data and make predicti

# Grid search to find the best ARIMA parameters
def arima_parameters(df, feature, search_range=2):
    """Grid search for the optimal parameters of the Arima model for
    :param: df, pdf.DataFrame, data
    :param: feature, str, feature name.
    :param: search_range, int, the range for the search of the param
    """
    p = d = q = range(0, search_range)
```

```
    p = d = q = range(0, search_range)
    pdq = list(itertools.product(p, d, q))
    seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.pr

    minimal_aic = 0
    optimal_param =[]
    for param in pdq:
        for param_seasonal in seasonal_pdq:
            try:
                mod = sm.tsa.statespace.SARIMAX(df[feature].resample
                results = mod.fit()
                print('ARIMA{}x{}12 - AIC:{}'.format(param, param_sea
                if results.aic < minimal_aic:
                    optimal_param = [param, param_seasonal]
                    minimal_aic = results.aic
                    print(minimal_aic)
            except:
                continue
    print('\n Optimal parameters ARIMA{}x{}12 - Minimal AIC:{}'.forma
    return optimal_param[0], optimal_param[1]



def arima_train(df, feature):
    """Train the arima model with the optimal parameters computed fo
    """
    order, seasonal_order =  arima_parameters(df, feature)
    mod = sm.tsa.statespace.SARIMAX(df[feature].resample('MS').mean(
                                order=order,
                                seasonal_order=seasonal_order,
                                enforce_stationarity=False,
                                enforce_invertibility=False)
    results = mod.fit()
    return results

def arima_diagonostics(results):
    results.plot_diagnostics(figsize=(16, 8))
    plt.show()

def arima_table(results):
    print(results.summary().tables[1])

def arima_predict(results, df, feature, init_date = "2009-01-01", sta
    pred = results.get_prediction(start=pd.to_datetime(start_date), 
    pred_ci = pred.conf_int()
    y = df[feature].resample("MS").mean()
    ax = y[init_date:].plot(label='observed')
    pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', 
    ax.fill_between(pred_ci.index,
                    pred_ci.iloc[:, 0],
                    pred_ci.iloc[:, 1], color='k', alpha=.2)
    ax.set_xlabel('Date')
    ax.set_ylabel(feature)
    plt.legend()
    plt.show()
    y_forecasted = pred.predicted_mean
```

```
    y_truth = y['2012-01-01':]
    mse = ((y_forecasted - y_truth) ** 2).mean()
    print('The Mean Squared Error of our forecasts is {}\n'.format(r
    print('The Root Mean Squared Error of our forecasts is {}'.forma


def arima_forcast(results, df, feature):
    pred_uc = results.get_forecast(steps=100)
    pred_ci = pred_uc.conf_int()
    y = df[feature].resample("MS").mean()
    ax = y.plot(label='observed', figsize=(14, 7))
    pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
    ax.fill_between(pred_ci.index,
                    pred_ci.iloc[:, 0],
                    pred_ci.iloc[:, 1], color='k', alpha=.25)
    ax.set_xlabel('Date')
    ax.set_ylabel(feature)
    plt.legend()
    plt.show()
```

```
#@title Example: decompose "Consumption" column into trend, seasonal

# replace "Consumption" with any feature in our data to get more plo

plot_component(df, "Consumption")
```

Example: deco
seasonal and



Time series analysis with `ARIMA`

```
#@title Grid search for optimal ```ARIMA``` parameters

# We find the optimal parameters for "Inflation": ARIMA(1, 1, 1)x(0,
arima_parameters(df, "Inflation")
```

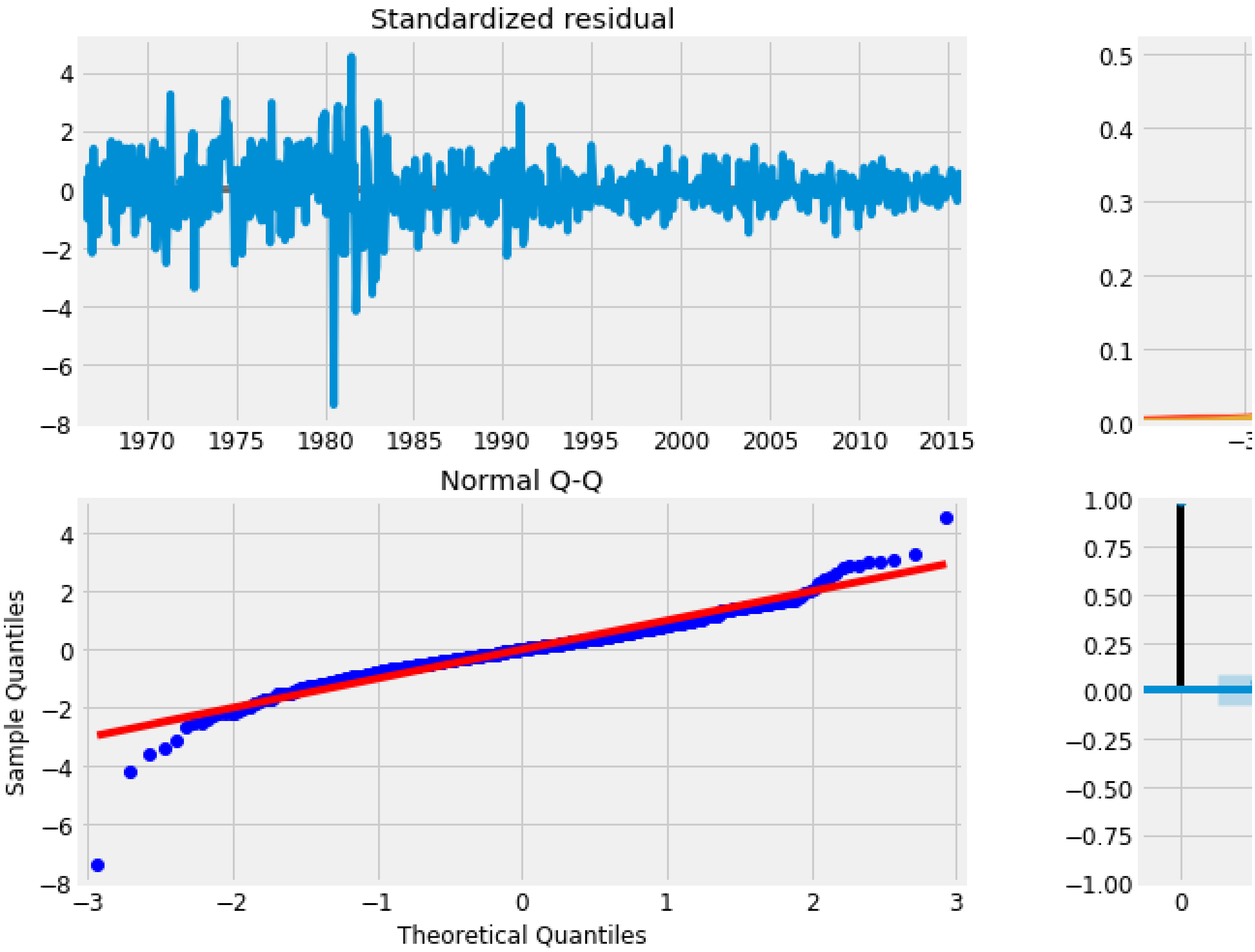
Grid search fo

```
#@title ```ARIMA``` training

results = arima_train(df, "Inflation")
```

ARIMA training

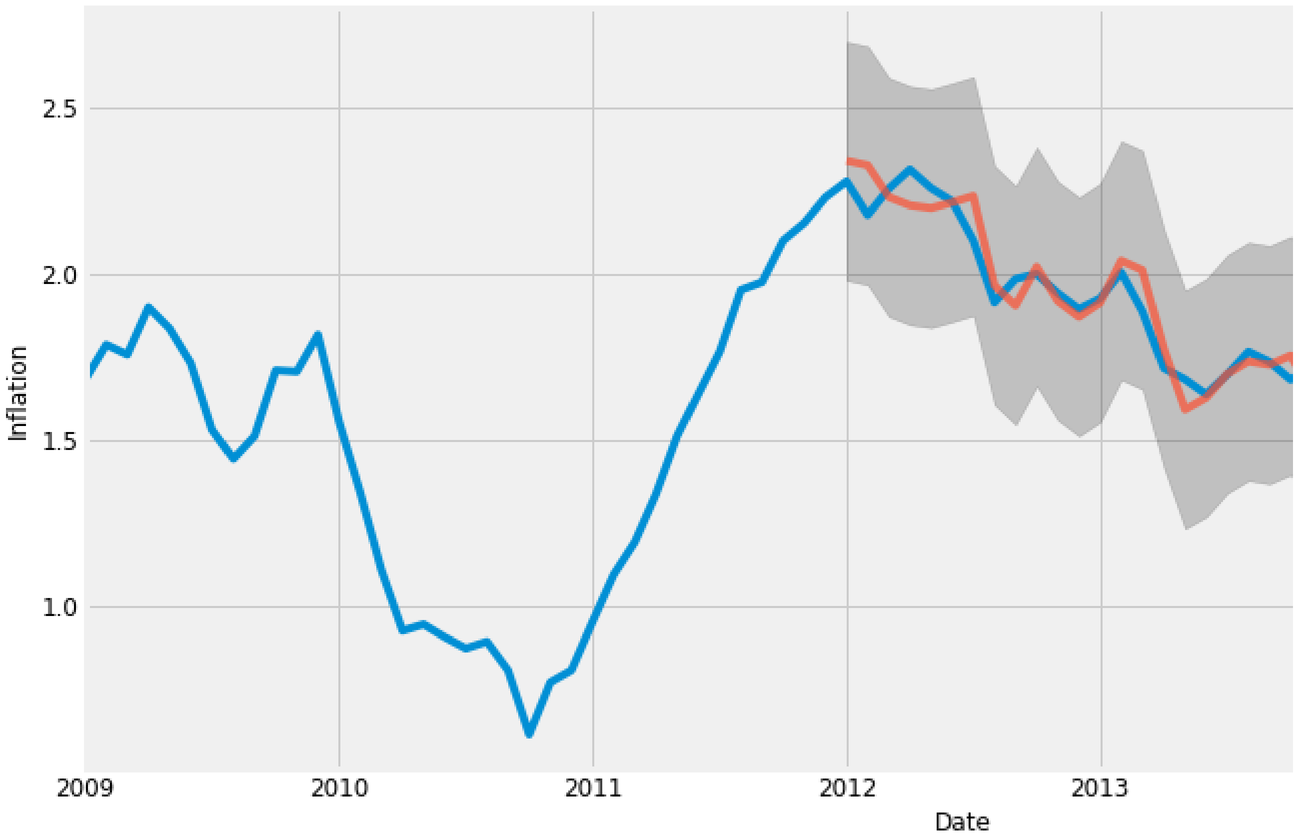
```
#@title ```ARIMA``` diadonostics
arima_diagonostics(results)
```

ARIMA diadon




```
#@title ```ARIMA``` predictions
arima_predict(results, df, "Inflation")
```

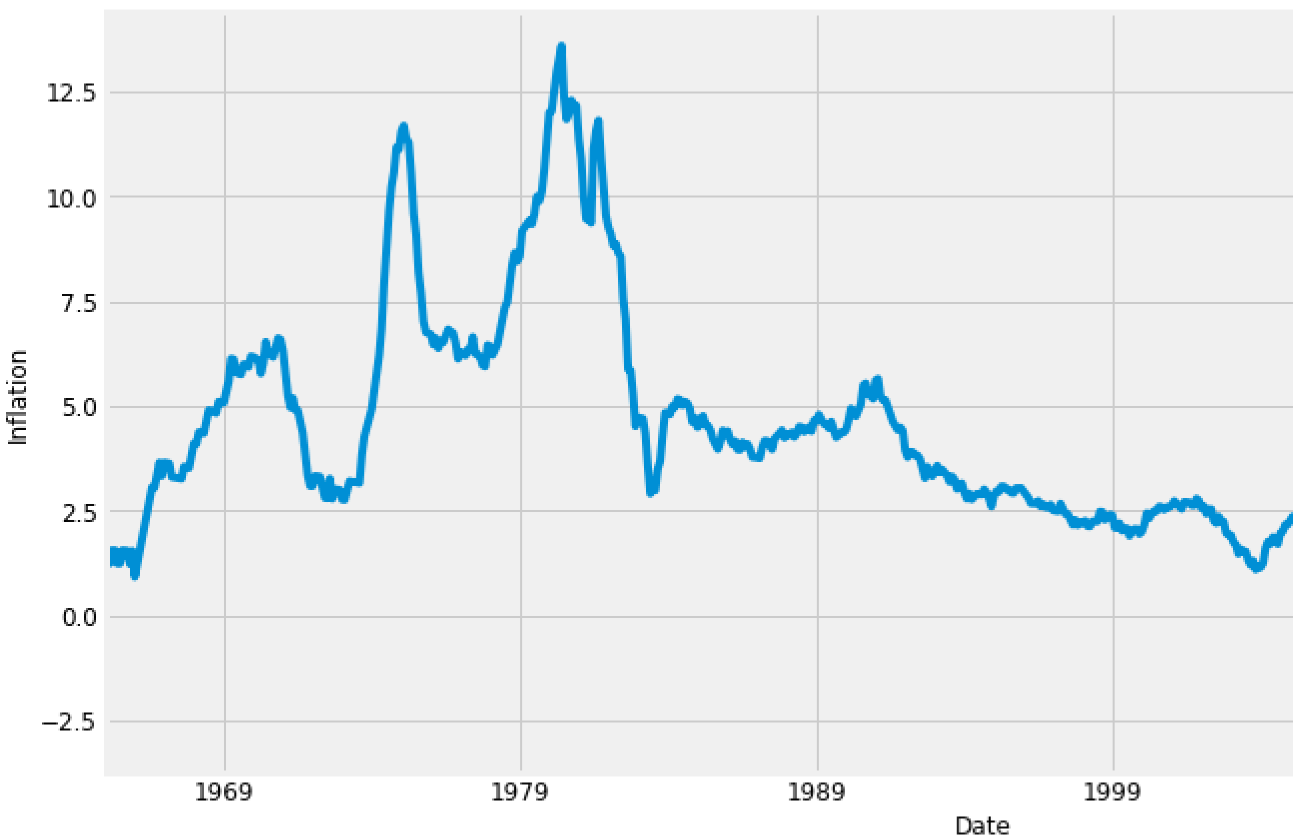
ARIMA predicti




```
The Mean Squared Error of our forecasts is 0.004963826636415743

The Root Mean Squared Error of our forecasts is 0.07
```

```
#@title ```ARIMA``` forcasts
arima_forcast(results, df, "Inflation")
```

ARIMA forcast



Data Analysis

- Components plot show the obvious seasonality, for example, in every 10 years, the "**Inflati**
  a **half-year seasonality**.
- The optimal ARIMA parameters for "Inflation" are `(1, 1, 1)x(0, 0, 1, 12)`
- The ARIMA diagonostics show that the **noise distribution is narrower than the**
- **The one-step ahead forcast captures the overall trend well**.
- As we forecast further out into the future, we becomes less confident in our values. This is r
  by our model, which grow larger as we move further out into the future.

Part II.1 Basic model: single-step, single-feature fo

Recurrent Neural Networks (RNNs) are good fits for time-series analysis because R
designed to capture patterns developing through time.

However, vanilla RNNs have a major disadvantage---the vanishing gradient problem---"the changes
so small, making the network unable to converge to a optimal solution.

LSTM (Long-Short Term Memory) is a variation of vanilla RNNS,it overcomes the va
problem by clipping gradients if they exceed some constant bounds.

In this section, we will

- Process the data to fit the LSTM model
- **Build and train the LSTM model for single-step, single-feature pred**
  tomorrow value with only today's values of the other 5 features).

```
#@title imports
import math
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import LSTM, Dense
```

imports

```
#@title Data preparation

def transform_data(df, features, targets, look_back = 0, look_forwar
  """transform the data in a custom form.

  :param: df, pd.DataFram, the data,
    e.g. df = pd.read_excel("USMacroData.xls", "All")
  :param: features, list of strs, the features to be uses as the sou
     e.g. ["Wage", "Consumption"]
  :param: look_back, int, number of days to look back in historic da
     e.g. look_back = 11 means we use the last (11+1)=12 months' dat
  :param:look_forward, int, num of days to look forward
    e.g. look_forward = 3 means we want to predict next 3 months' da
  :param: split_ratio, float, split the data into training dataset a
```

Data preparati

```
      e.g. split_ratio=0.7 means we use the first 70% of the data as tr
    :rtype: np.arrays, x_train, y_train, x_test, y_test
    """
    x, y = [], []
    for i in range(look_back, len(df) - look_forward):
        assert look_back < len(df)-look_forward, "Invalid look_back, lo

        x.append(np.array(df[i-look_back : i+1][features]))
        y.append(np.array(df[i+1: i+look_forward+1][targets]).transpose

    # List to np.arrary
    x_arr = np.array(x)
    y_arr = np.array(y)

    split_point = int(len(x)*split_ratio)

    return x_arr[0:split_point], y_arr[0:split_point], x_arr[split_poi

features = ["Wage", "Unemployment", "Consumption", "Investment", "In
targets = ["Inflation"]
x_train, y_train, x_test, y_test = transform_data(df, features=featu

#Note that all returned np.arrays are three dimensional.
#Need to reshape y_train and y_test to fit the LSTM

# For the basic model only
y_train = np.reshape(y_train, (y_train.shape[0], -1))
y_test = np.reshape(y_test, (y_test.shape[0], -1))
```

```
#@title Build and train the LSTM model

# To match the Input shape (1,5) and our x_train shape is very impor

def train_model(Optimizer, x_train, y_train, x_test, y_test):
  model = Sequential()
  model.add(LSTM(50, input_shape=(1, 5)))
  model.add(Dense(1))

  model.compile(loss="mean_squared_error", optimizer=Optimizer, metr
  scores = model.fit(x=x_train,y=y_train, batch_size=1, epochs = 100

  return scores, model
```

Build and trair

```
#@title Make sure data forms are correct
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

Make sure dat

```
(427, 1, 5)
(427, 1)
(184, 1, 5)
(184, 1)
```

```
#@title LSTM with SGD, RMSprop, Adam optimizers, epochs = 100
#SGD_score, SGD_model = train_model(Optimizer = "sgd", x_train=x_tra
RMSprop_score, RMSprop_model = train_model(Optimizer = "RMSprop", x_
#Adam_score, Adam_model = train_model(Optimizer = "adam", x_train=x_
```

LSTM with SG
100

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Train on 427 samples, validate on 184 samples
Epoch 1/100
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

427/427 [==============================] - 2s 6ms/step - loss: 6.8603 - acc: 0.0000e+
Epoch 2/100
427/427 [==============================] - 1s 3ms/step - loss: 2.4877 - acc: 0.0000e+
Epoch 3/100
427/427 [==============================] - 1s 3ms/step - loss: 1.8379 - acc: 0.0000e+
Epoch 4/100
427/427 [==============================] - 1s 3ms/step - loss: 1.4932 - acc: 0.0000e+
Epoch 5/100
427/427 [==============================] - 1s 3ms/step - loss: 1.2980 - acc: 0.0000e+
Epoch 6/100
427/427 [==============================] - 1s 3ms/step - loss: 1.2265 - acc: 0.0000e+
Epoch 7/100
427/427 [==============================] - 1s 3ms/step - loss: 1.0970 - acc: 0.0000e+
Epoch 8/100
427/427 [==============================] - 1s 3ms/step - loss: 1.0660 - acc: 0.0000e+
Epoch 9/100
427/427 [==============================] - 1s 3ms/step - loss: 1.0541 - acc: 0.0000e+
Epoch 10/100
427/427 [==============================] - 1s 3ms/step - loss: 1.0670 - acc: 0.0000e+
Epoch 11/100
427/427 [==============================] - 1s 3ms/step - loss: 1.0320 - acc: 0.0000e+
Epoch 12/100
427/427 [==============================] - 1s 3ms/step - loss: 1.0103 - acc: 0.0000e+
Epoch 13/100
427/427 [==============================] - 1s 3ms/step - loss: 0.9960 - acc: 0.0000e+
Epoch 14/100
427/427 [==============================] - 1s 3ms/step - loss: 0.9939 - acc: 0.0000e+
Epoch 15/100
427/427 [==============================] - 1s 3ms/step - loss: 0.9544 - acc: 0.0000e+
Epoch 16/100
427/427 [==============================] - 1s 3ms/step - loss: 0.9127 - acc: 0.0000e+
Epoch 17/100
```

```
427/427 [==============================] - 1s 3ms/step - loss: 0.9417 - acc: 0.0000e+
Epoch 18/100
427/427 [==============================] - 1s 3ms/step - loss: 0.9365 - acc: 0.0000e+
Epoch 19/100
427/427 [==============================] - 1s 3ms/step - loss: 0.9319 - acc: 0.0000e+
Epoch 20/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8785 - acc: 0.0000e+
Epoch 21/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8815 - acc: 0.0000e+
Epoch 22/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8928 - acc: 0.0000e+
Epoch 23/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8889 - acc: 0.0000e+
Epoch 24/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8879 - acc: 0.0000e+
Epoch 25/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8620 - acc: 0.0000e+
Epoch 26/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8465 - acc: 0.0000e+
Epoch 27/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8588 - acc: 0.0000e+
Epoch 28/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8365 - acc: 0.0000e+
Epoch 29/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8479 - acc: 0.0000e+
Epoch 30/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8323 - acc: 0.0000e+
Epoch 31/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7964 - acc: 0.0000e+
Epoch 32/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8261 - acc: 0.0000e+
Epoch 33/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8114 - acc: 0.0000e+
Epoch 34/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8281 - acc: 0.0000e+
Epoch 35/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7679 - acc: 0.0000e+
Epoch 36/100
427/427 [==============================] - 1s 3ms/step - loss: 0.8105 - acc: 0.0000e+
Epoch 37/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7924 - acc: 0.0000e+
Epoch 38/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7719 - acc: 0.0000e+
Epoch 39/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7930 - acc: 0.0000e+
Epoch 40/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7406 - acc: 0.0000e+
Epoch 41/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7874 - acc: 0.0000e+
Epoch 42/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7816 - acc: 0.0000e+
Epoch 43/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7781 - acc: 0.0000e+
Epoch 44/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7825 - acc: 0.0000e+
Epoch 45/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7498 - acc: 0.0000e+
Epoch 46/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7355 - acc: 0.0000e+
Epoch 47/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7383 - acc: 0.0000e+
Epoch 48/100
```

```
Epoch 48/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7791 - acc: 0.0000e+
Epoch 49/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7407 - acc: 0.0000e+
Epoch 50/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7473 - acc: 0.0000e+
Epoch 51/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7301 - acc: 0.0000e+
Epoch 52/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7511 - acc: 0.0000e+
Epoch 53/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6949 - acc: 0.0000e+
Epoch 54/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7170 - acc: 0.0000e+
Epoch 55/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7276 - acc: 0.0000e+
Epoch 56/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7010 - acc: 0.0000e+
Epoch 57/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7188 - acc: 0.0000e+
Epoch 58/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7016 - acc: 0.0000e+
Epoch 59/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7295 - acc: 0.0000e+
Epoch 60/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7264 - acc: 0.0000e+
Epoch 61/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6682 - acc: 0.0000e+
Epoch 62/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7222 - acc: 0.0000e+
Epoch 63/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6951 - acc: 0.0000e+
Epoch 64/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7004 - acc: 0.0000e+
Epoch 65/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7102 - acc: 0.0000e+
Epoch 66/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7045 - acc: 0.0000e+
Epoch 67/100
427/427 [==============================] - 1s 3ms/step - loss: 0.7047 - acc: 0.0000e+
Epoch 68/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6977 - acc: 0.0000e+
Epoch 69/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6832 - acc: 0.0000e+
Epoch 70/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6627 - acc: 0.0000e+
Epoch 71/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6873 - acc: 0.0000e+
Epoch 72/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6946 - acc: 0.0000e+
Epoch 73/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6873 - acc: 0.0000e+
Epoch 74/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6566 - acc: 0.0000e+
Epoch 75/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6901 - acc: 0.0000e+
Epoch 76/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6839 - acc: 0.0000e+
Epoch 77/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6527 - acc: 0.0000e+
Epoch 78/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6459 - acc: 0.0000e+
```

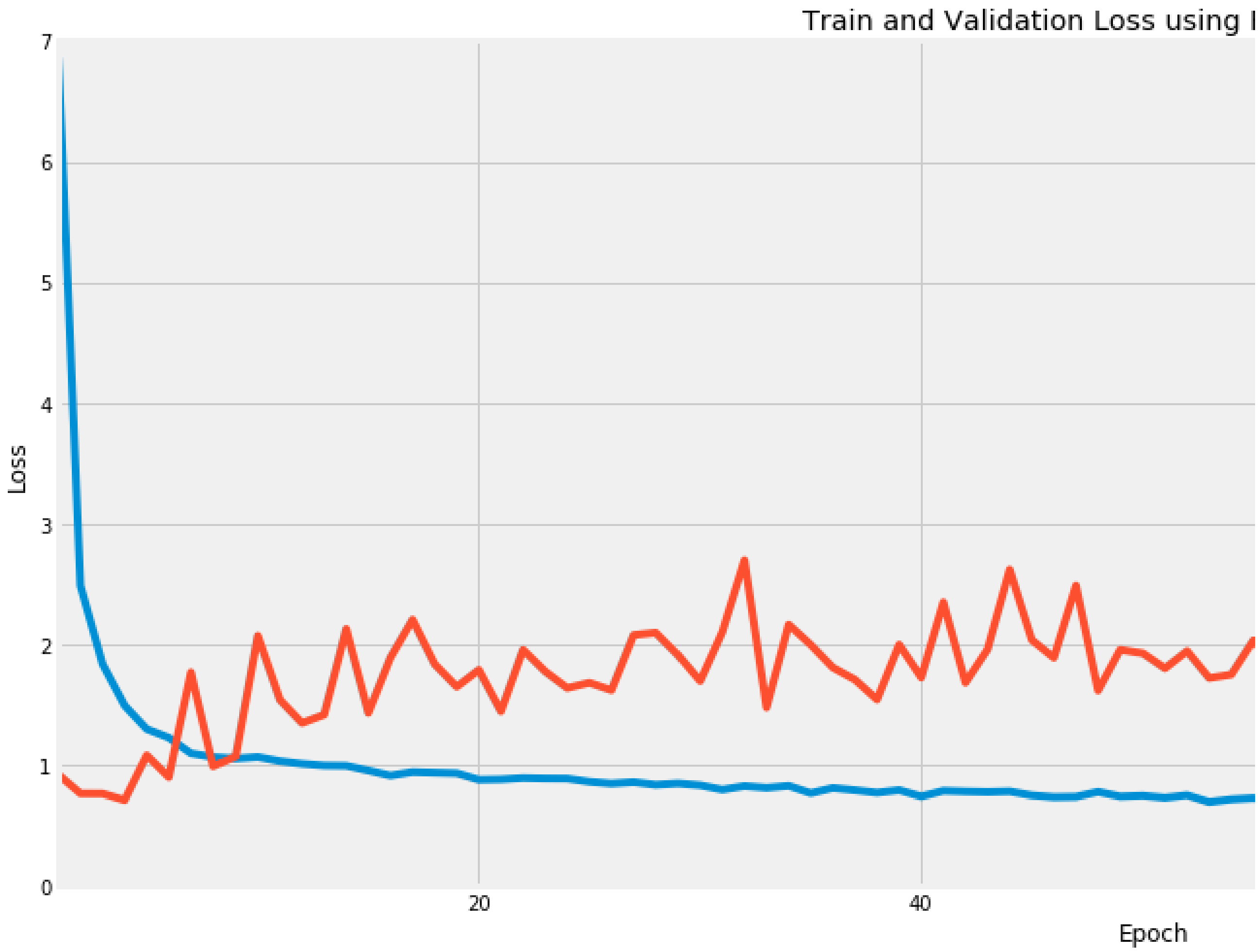
```
Epoch 79/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6751 - acc: 0.0000e+
Epoch 80/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6639 - acc: 0.0000e+
Epoch 81/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6555 - acc: 0.0000e+
Epoch 82/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6601 - acc: 0.0000e+
Epoch 83/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6115 - acc: 0.0000e+
Epoch 84/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6476 - acc: 0.0000e+
Epoch 85/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6256 - acc: 0.0000e+
Epoch 86/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6768 - acc: 0.0000e+
Epoch 87/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6399 - acc: 0.0000e+
Epoch 88/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6266 - acc: 0.0000e+
Epoch 89/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6240 - acc: 0.0000e+
Epoch 90/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6269 - acc: 0.0000e+
Epoch 91/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6289 - acc: 0.0000e+
Epoch 92/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6331 - acc: 0.0000e+
Epoch 93/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6190 - acc: 0.0000e+
Epoch 94/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6248 - acc: 0.0000e+
Epoch 95/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6257 - acc: 0.0000e+
Epoch 96/100
427/427 [==============================] - 1s 3ms/step - loss: 0.6182 - acc: 0.0000e+
Epoch 97/100
427/427 [==============================] - 1s 3ms/step - loss: 0.5983 - acc: 0.0000e+
Epoch 98/100
427/427 [==============================] - 1s 3ms/step - loss: 0.5794 - acc: 0.0000e+
Epoch 99/100
427/427 [==============================] - 1s 3ms/step - loss: 0.5872 - acc: 0.0000e+
Epoch 100/100
```

Plot result

```
#@title Plot result

def plot_result(score, optimizer_name, label = "loss"):
  plt.figure(figsize=(18, 8))
  plt.plot(range(1, 101), score.history["loss"], label ="Training Los
  plt.plot(range(1,101), score.history["val_loss"], label="Validation
  plt.axis([1,100, 0, 7])
  plt.xlabel("Epoch")
  plt.ylabel("Loss")
  plt.title("Train and Validation Loss using "+optimizer_name + "opt
  plt.legend()
  plt.show()
```

```
#@title Plot result
plot_result(RMSprop_score, "RMSprop")
```

Plot result




```
#@title Plot predictions
def plot_predict(model, x_train, x_test, y_train, y_test):
  train_predict = RMSprop_model.predict(x_train)
  test_predict = RMSprop_model.predict(x_test)

  # Calculate root mean squared error.
  trainScore = math.sqrt(mean_squared_error(y_train, train_predict))
  print('Train Score: %.2f RMSE' % (trainScore))
  testScore = math.sqrt(mean_squared_error(y_test, test_predict))
  print('Test Score: %.2f RMSE' % (testScore))

  plt.figure(figsize=(18, 8))
  plt.plot(train_predict)
  plt.plot(y_train)
  plt.show()

  plt.figure(figsize=(18, 8))
  plt.plot(test_predict)
  plt.plot(y_test)
  plt.show()
```

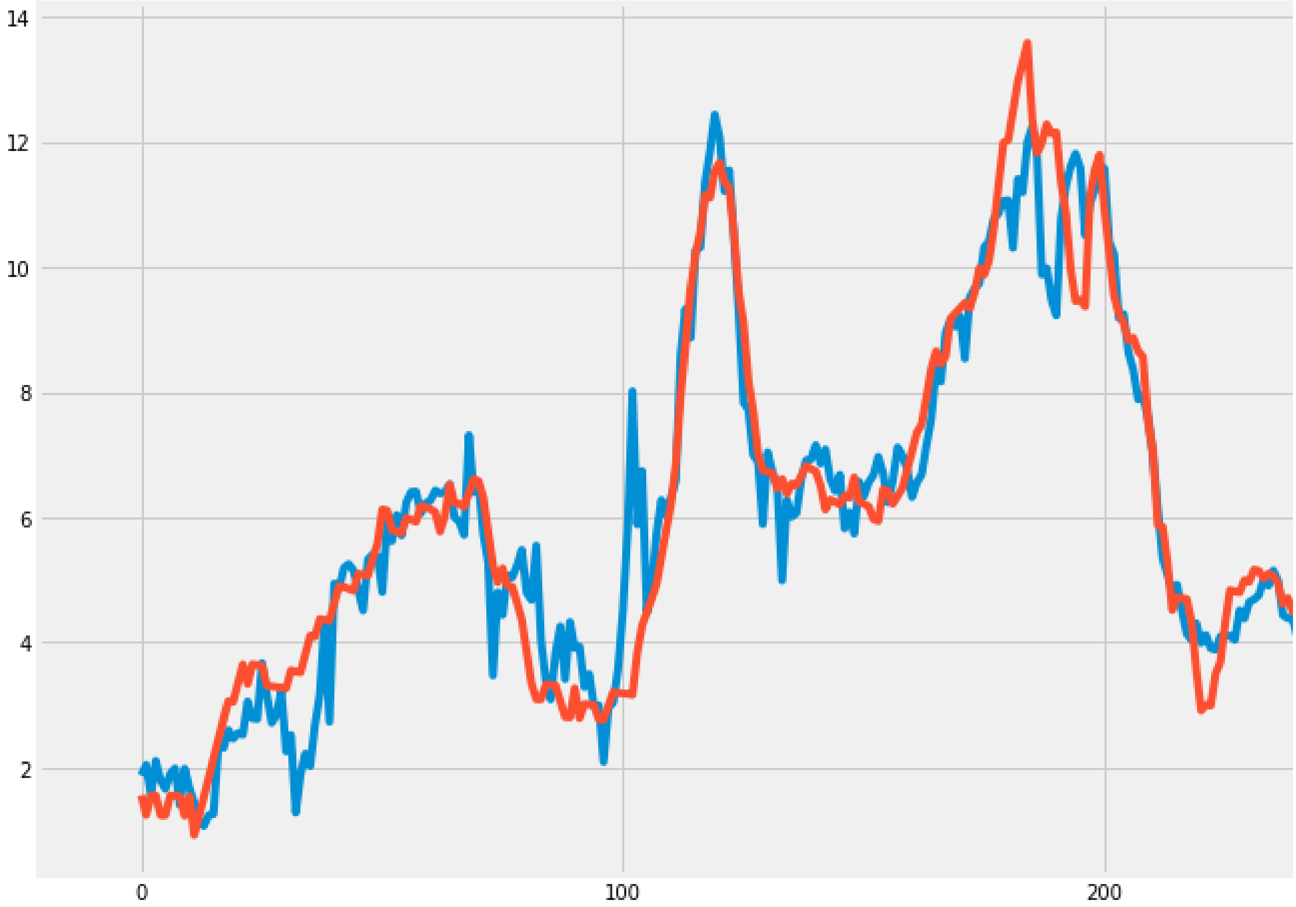
Plot predictior

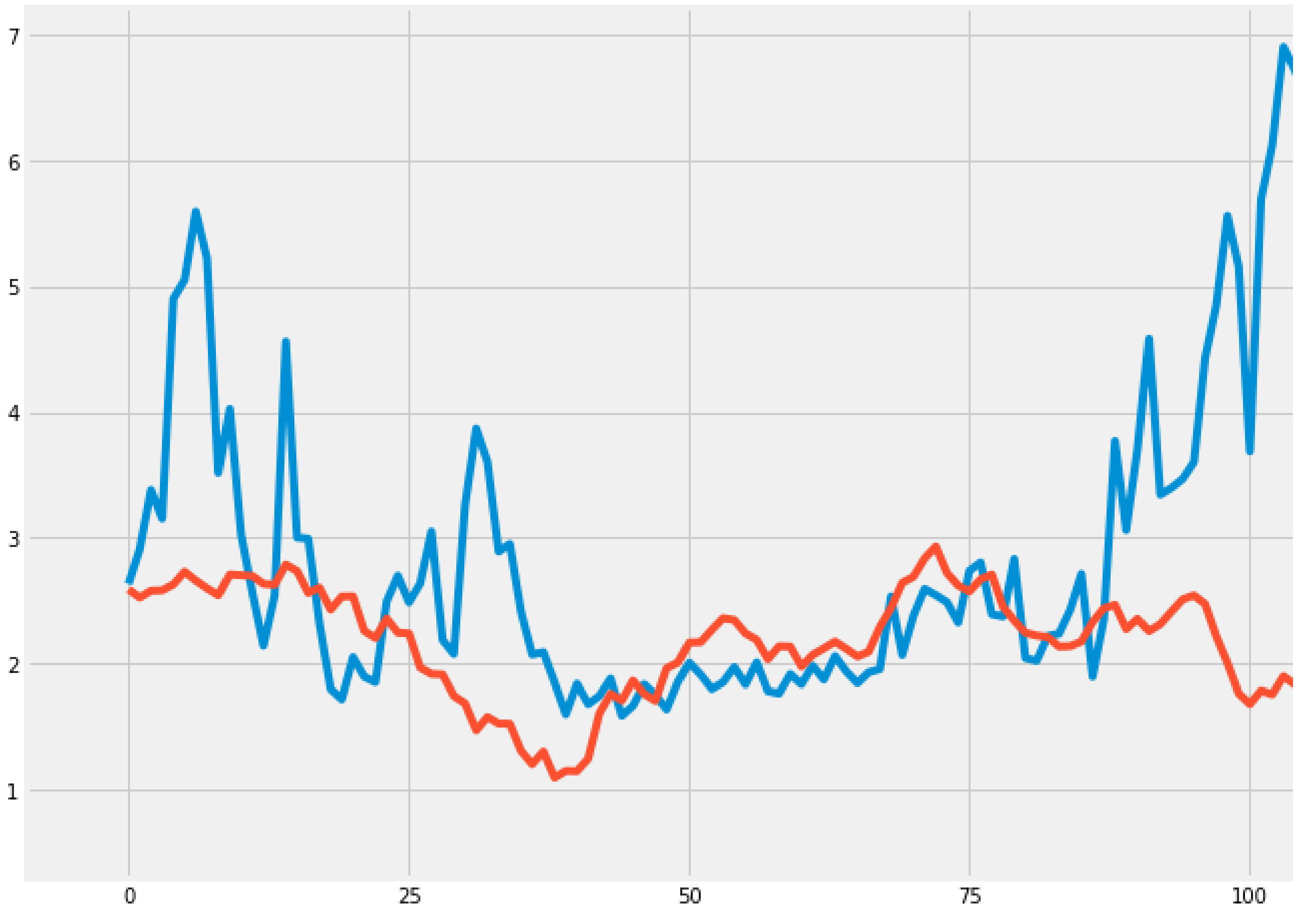
```
#@title Plot predictions
```

Plot predictior

```
plot_predict(RMSprop_model, x_train = x_train, y_train=y_train, x_te
```

```
Train Score: 0.71 RMSE
Test Score: 1.53 RMSE
```





Data Analysis

We only trained the model for 100 epochs, feel free to modify it to any number as long as we have
results we find during the experiments

- LSTM with Adam or RMSprop optimizers work better than the SGD optimizer in this project.
- Each model fits the training dataset very well.
- **The prediction captures the range and characteristics of the real dat**
- **The model doesn't predict the rapid increasing near the 100th test d**

Part II.2 Generalized model: multi-step, multi-fea

We build a multi-step, multi-feature LSTM model in this section. That means we can use several-d
features in the future.
For example, we can use last 12-month's data of Wage, Consumption, Ir
. In this section, we

- Process the data to fit the requirements of all possible multi-step, multi-feature prediction ta
- We modify the LSTM model accordingly.
- Plot the 3-month prediction for Inflation and Unemployment with last 12-month's data of Wa
  InterestRate.

```
#@title Data preparation

def transform_data(df, features, targets, look_back = 0, look_forwar
  """transform the data in a custom form.

  :param: df, pd.DataFram, the data,
    e.g. df = pd.read_excel("USMacroData.xls", "All")
  :param: features, list of strs, the features to be uses as the sou
     e.g. ["Wage", "Consumption"]
  :param: look_back, int, number of days to look back in historic da
     e.g. look_back = 11 means we use the last (11+1)=12 months' dat
  :param:look_forward, int, num of days to look forward
    e.g. look_forward = 3 means we want to predict next 3 months' da
  :param: split_ratio, float, split the data into training dataset a
    e.g. split_ratio=0.7 means we use the first 70% of the data as t
  :rtype: np.arrays, x_train, y_train, x_test, y_test
  """
  x, y = [], []
  for i in range(look_back, len(df) - look_forward):
      assert look_back < len(df)-look_forward, "Invalid look_back, l

      x.append(np.array(df[i-look_back : i+1][features]))
      y.append(np.array(df[i+1: i+look_forward+1][targets]).transpos

  # List to np.arrary
  x_arr = np.array(x)
  y_arr = np.array(y)
```

Data preparati

```
  split_point = int(len(x)*split_ratio)

  return x_arr[0:split_point], y_arr[0:split_point], x_arr[split_poi

features = ["Wage", "Consumption", "Investment", "InterestRate"]
targets = ["Inflation", "Unemployment"]
x_train, y_train, x_test, y_test = transform_data(df, features=featu

#Note that all returned np.arrays are three dimensional.
#Need to reshape y_train and y_test to fit the LSTM

# For the multi-step LSTM model only
y_train = np.reshape(y_train, (y_train.shape[0], -1))
y_test = np.reshape(y_test, (y_test.shape[0], -1))
```

```
#@title Make the data forms are all correct
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_train.shape)
```

Make the data

```
(418, 12, 4)
(418, 6)
(180, 12, 4)
(418, 6)
```

```
#@title Scaling, vectorize and de_vectorize

def scale(arr, df):
  """Scale the data to range (-1,1) to better fit the LSTM model

  :param: arr, np.array, the array to be scaled
  :param: df, pd.DataFrame, to provide the max and min for us to sca
  TODO: maybe we don't need the df parameter?
  """
  global_max = max(df.max())
  global_min = min(df.min())
  arr = -1 + (arr-global_min)*2/(global_max-global_min)
  return arr

def de_scale(arr, df):
  """de Scale the data from range (-1,1) to its original range

  :param: arr, np.array, the array to be scaled
  :param: df, pd.DataFrame, to provide the max and min for us to sca
  """
  global_max = max(df.max())
  global_min = min(df.min())
  arr = global_min+(arr+1)*(global_max-global_min)/2
  return arr

def vectorize(y_train):
  """To vectorize an np.array.
```

Scaling, vecto

```
  :param: y_train, np.array, the array to be vectorized
  :rtype: np.array, vectorized array.
  """
  return np.reshape(y_train, (y_train.shape[0], -1))

def de_vectorize(y_train, row, col):
  """To de_vectorize an np.array: transfrom from 2-dim np.array to i

  :param: y_train, np.array, the array to be de_vectorized
  :rtype: np.array, de_vectorized array.
  """
  return np.reshape(y_train,(y_train.shape[0], row, col))
```

```
#@title Multi-step LSTM model, change the input_shape and Dense layer

def train_multi_step_model(Optimizer, x_train, y_train, x_test, y_te
  model = Sequential()
  model.add(LSTM(50, input_shape=(12, 4)))
  model.add(Dense(6))

  model.compile(loss="mean_squared_error", optimizer=Optimizer, metr
  scores = model.fit(x=x_train,y=y_train, batch_size=1, epochs = 200

  return scores, model
```

Multi-step LST
Dense layer pa
test data shap

```
#@title Train the model. Change the optimizer parameter to use other
RMS_score, RMS_model = train_multi_step_model(Optimizer = "RMSprop",
```

Train the mod
other optimize

```
Train on 418 samples, validate on 180 samples
Epoch 1/200
418/418 [==============================] - 5s 12ms/step - loss: 5.9280 - acc: 0.2895
Epoch 2/200
418/418 [==============================] - 5s 11ms/step - loss: 1.9184 - acc: 0.4043
Epoch 3/200
418/418 [==============================] - 4s 11ms/step - loss: 1.4018 - acc: 0.3541
Epoch 4/200
418/418 [==============================] - 4s 11ms/step - loss: 1.1442 - acc: 0.3373
Epoch 5/200
418/418 [==============================] - 4s 11ms/step - loss: 0.9653 - acc: 0.3493
Epoch 6/200
418/418 [==============================] - 4s 10ms/step - loss: 0.8703 - acc: 0.3876
Epoch 7/200
418/418 [==============================] - 4s 11ms/step - loss: 0.7075 - acc: 0.3684
Epoch 8/200
418/418 [==============================] - 4s 10ms/step - loss: 0.6512 - acc: 0.3923
Epoch 9/200
418/418 [==============================] - 4s 10ms/step - loss: 0.5565 - acc: 0.4163
Epoch 10/200
418/418 [==============================] - 4s 11ms/step - loss: 0.4753 - acc: 0.4354
Epoch 11/200
418/418 [==============================] - 4s 11ms/step - loss: 0.4950 - acc: 0.4139
Epoch 12/200
418/418 [==============================] - 5s 11ms/step - loss: 0.4238 - acc: 0.4474
Epoch 13/200
418/418 [==============================] - 5s 11ms/step - loss: 0.4073 - acc: 0.4234
Epoch 14/200
418/418 [==============================] - 5s 11ms/step - loss: 0.3882 - acc: 0.4665
Epoch 15/200
418/418 [==============================] - 5s 11ms/step - loss: 0.3522 - acc: 0.4354
Epoch 16/200
418/418 [==============================] - 5s 12ms/step - loss: 0.3426 - acc: 0.4354
Epoch 17/200
418/418 [==============================] - 5s 11ms/step - loss: 0.3134 - acc: 0.4617
Epoch 18/200
418/418 [==============================] - 5s 11ms/step - loss: 0.2790 - acc: 0.4665
Epoch 19/200
418/418 [==============================] - 5s 11ms/step - loss: 0.2807 - acc: 0.4545
Epoch 20/200
418/418 [==============================] - 5s 11ms/step - loss: 0.2760 - acc: 0.4809
Epoch 21/200
418/418 [==============================] - 4s 11ms/step - loss: 0.2589 - acc: 0.5048
Epoch 22/200
418/418 [==============================] - 5s 11ms/step - loss: 0.2368 - acc: 0.4809
Epoch 23/200
418/418 [==============================] - 5s 11ms/step - loss: 0.2378 - acc: 0.4737
Epoch 24/200
418/418 [==============================] - 4s 11ms/step - loss: 0.2329 - acc: 0.4761
Epoch 25/200
418/418 [==============================] - 5s 11ms/step - loss: 0.2324 - acc: 0.5191
Epoch 26/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1927 - acc: 0.4904
Epoch 27/200
418/418 [==============================] - 5s 11ms/step - loss: 0.2153 - acc: 0.5096
Epoch 28/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1898 - acc: 0.5120
Epoch 29/200
418/418 [==============================] - 5s 12ms/step - loss: 0.1831 - acc: 0.4976
Epoch 30/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1925 - acc: 0.5144
```

```
Epoch 31/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1713 - acc: 0.5167
Epoch 32/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1675 - acc: 0.5024
Epoch 33/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1622 - acc: 0.5000
Epoch 34/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1885 - acc: 0.4928
Epoch 35/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1623 - acc: 0.4689
Epoch 36/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1560 - acc: 0.5167
Epoch 37/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1556 - acc: 0.5431
Epoch 38/200
418/418 [==============================] - 5s 12ms/step - loss: 0.1548 - acc: 0.5072
Epoch 39/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1414 - acc: 0.5431
Epoch 40/200
418/418 [==============================] - 5s 12ms/step - loss: 0.1353 - acc: 0.5191
Epoch 41/200
418/418 [==============================] - 5s 12ms/step - loss: 0.1457 - acc: 0.5072
Epoch 42/200
418/418 [==============================] - 5s 12ms/step - loss: 0.1308 - acc: 0.5191
Epoch 43/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1170 - acc: 0.5048
Epoch 44/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1353 - acc: 0.5024
Epoch 45/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1350 - acc: 0.5287
Epoch 46/200
418/418 [==============================] - 5s 12ms/step - loss: 0.1127 - acc: 0.5215
Epoch 47/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1181 - acc: 0.5311
Epoch 48/200
418/418 [==============================] - 5s 12ms/step - loss: 0.1288 - acc: 0.5239
Epoch 49/200
418/418 [==============================] - 4s 10ms/step - loss: 0.1176 - acc: 0.5239
Epoch 50/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1106 - acc: 0.5215
Epoch 51/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1047 - acc: 0.5144
Epoch 52/200
418/418 [==============================] - 5s 11ms/step - loss: 0.1037 - acc: 0.5502
Epoch 53/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0972 - acc: 0.5096
Epoch 54/200
418/418 [==============================] - 4s 11ms/step - loss: 0.1043 - acc: 0.5550
Epoch 55/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0986 - acc: 0.5287
Epoch 56/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0971 - acc: 0.5694
Epoch 57/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0981 - acc: 0.5335
Epoch 58/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0982 - acc: 0.5335
Epoch 59/200
418/418 [==============================] - 5s 12ms/step - loss: 0.0952 - acc: 0.5431
Epoch 60/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0990 - acc: 0.5263
Epoch 61/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0919 - acc: 0.5335
```

```
Epoch 62/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0878 - acc: 0.5311
Epoch 63/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0898 - acc: 0.5502
Epoch 64/200
418/418 [==============================] - 5s 12ms/step - loss: 0.1000 - acc: 0.5167
Epoch 65/200
418/418 [==============================] - 5s 12ms/step - loss: 0.0819 - acc: 0.5526
Epoch 66/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0896 - acc: 0.5718
Epoch 67/200
418/418 [==============================] - 5s 12ms/step - loss: 0.0902 - acc: 0.5431
Epoch 68/200
418/418 [==============================] - 5s 12ms/step - loss: 0.0774 - acc: 0.5024
Epoch 69/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0836 - acc: 0.5311
Epoch 70/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0901 - acc: 0.5526
Epoch 71/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0884 - acc: 0.5407
Epoch 72/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0970 - acc: 0.5359
Epoch 73/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0739 - acc: 0.5694
Epoch 74/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0730 - acc: 0.5598
Epoch 75/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0807 - acc: 0.5335
Epoch 76/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0757 - acc: 0.5502
Epoch 77/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0856 - acc: 0.5478
Epoch 78/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0697 - acc: 0.5526
Epoch 79/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0659 - acc: 0.5574
Epoch 80/200
418/418 [==============================] - 5s 12ms/step - loss: 0.0819 - acc: 0.5766
Epoch 81/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0737 - acc: 0.5191
Epoch 82/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0670 - acc: 0.5718
Epoch 83/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0702 - acc: 0.5574
Epoch 84/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0733 - acc: 0.5766
Epoch 85/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0701 - acc: 0.5718
Epoch 86/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0653 - acc: 0.5598
Epoch 87/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0727 - acc: 0.5431
Epoch 88/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0714 - acc: 0.5383
Epoch 89/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0722 - acc: 0.5478
Epoch 90/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0660 - acc: 0.5766
Epoch 91/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0625 - acc: 0.5718
Epoch 92/200
```

```
418/418 [==============================] - 5s 11ms/step - loss: 0.0623 - acc: 0.5813
Epoch 93/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0697 - acc: 0.5646
Epoch 94/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0656 - acc: 0.5335
Epoch 95/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0573 - acc: 0.5335
Epoch 96/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0591 - acc: 0.5191
Epoch 97/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0596 - acc: 0.5837
Epoch 98/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0561 - acc: 0.5598
Epoch 99/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0662 - acc: 0.5311
Epoch 100/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0594 - acc: 0.5383
Epoch 101/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0598 - acc: 0.5670
Epoch 102/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0594 - acc: 0.5191
Epoch 103/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0539 - acc: 0.5718
Epoch 104/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0580 - acc: 0.5550
Epoch 105/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0601 - acc: 0.5574
Epoch 106/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0552 - acc: 0.5502
Epoch 107/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0651 - acc: 0.5550
Epoch 108/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0549 - acc: 0.5598
Epoch 109/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0488 - acc: 0.5478
Epoch 110/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0501 - acc: 0.5431
Epoch 111/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0506 - acc: 0.5478
Epoch 112/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0516 - acc: 0.5335
Epoch 113/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0528 - acc: 0.5694
Epoch 114/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0548 - acc: 0.5574
Epoch 115/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0476 - acc: 0.5694
Epoch 116/200
418/418 [==============================] - 5s 13ms/step - loss: 0.0497 - acc: 0.5957
Epoch 117/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0511 - acc: 0.5478
Epoch 118/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0545 - acc: 0.5455
Epoch 119/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0605 - acc: 0.5646
Epoch 120/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0538 - acc: 0.5670
Epoch 121/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0523 - acc: 0.5455
Epoch 122/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0517 - acc: 0.5742
Epoch 123/200
```

```
Epoch 123/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0463 - acc: 0.5670
Epoch 124/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0461 - acc: 0.5813
Epoch 125/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0423 - acc: 0.5550
Epoch 126/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0479 - acc: 0.5550
Epoch 127/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0486 - acc: 0.5837
Epoch 128/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0541 - acc: 0.5407
Epoch 129/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0455 - acc: 0.5478
Epoch 130/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0582 - acc: 0.5670
Epoch 131/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0479 - acc: 0.5622
Epoch 132/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0408 - acc: 0.5622
Epoch 133/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0622 - acc: 0.5526
Epoch 134/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0443 - acc: 0.5383
Epoch 135/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0419 - acc: 0.5598
Epoch 136/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0398 - acc: 0.5431
Epoch 137/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0430 - acc: 0.5526
Epoch 138/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0430 - acc: 0.5646
Epoch 139/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0418 - acc: 0.5694
Epoch 140/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0429 - acc: 0.5813
Epoch 141/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0384 - acc: 0.5837
Epoch 142/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0419 - acc: 0.5670
Epoch 143/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0383 - acc: 0.5646
Epoch 144/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0431 - acc: 0.5909
Epoch 145/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0404 - acc: 0.5646
Epoch 146/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0421 - acc: 0.5502
Epoch 147/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0400 - acc: 0.5455
Epoch 148/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0397 - acc: 0.5550
Epoch 149/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0367 - acc: 0.5574
Epoch 150/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0366 - acc: 0.5359
Epoch 151/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0400 - acc: 0.5813
Epoch 152/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0369 - acc: 0.5455
Epoch 153/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0363 - acc: 0.5718
```

```
Epoch 154/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0405 - acc: 0.5455
Epoch 155/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0373 - acc: 0.5478
Epoch 156/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0429 - acc: 0.5455
Epoch 157/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0392 - acc: 0.5909
Epoch 158/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0395 - acc: 0.5646
Epoch 159/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0454 - acc: 0.5526
Epoch 160/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0428 - acc: 0.5335
Epoch 161/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0389 - acc: 0.5718
Epoch 162/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0451 - acc: 0.5574
Epoch 163/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0364 - acc: 0.5742
Epoch 164/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0412 - acc: 0.5598
Epoch 165/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0393 - acc: 0.5359
Epoch 166/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0408 - acc: 0.5646
Epoch 167/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0359 - acc: 0.5789
Epoch 168/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0431 - acc: 0.5742
Epoch 169/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0365 - acc: 0.5742
Epoch 170/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0400 - acc: 0.5646
Epoch 171/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0396 - acc: 0.5789
Epoch 172/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0393 - acc: 0.5718
Epoch 173/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0373 - acc: 0.5789
Epoch 174/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0394 - acc: 0.5909
Epoch 175/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0359 - acc: 0.5478
Epoch 176/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0361 - acc: 0.5670
Epoch 177/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0336 - acc: 0.5718
Epoch 178/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0420 - acc: 0.5766
Epoch 179/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0380 - acc: 0.5622
Epoch 180/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0358 - acc: 0.5502
Epoch 181/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0352 - acc: 0.5407
Epoch 182/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0404 - acc: 0.5885
Epoch 183/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0381 - acc: 0.6077
Epoch 184/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0359 - acc: 0.5718
```

```
Epoch 185/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0364 - acc: 0.5526
Epoch 186/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0351 - acc: 0.5742
Epoch 187/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0344 - acc: 0.5646
Epoch 188/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0348 - acc: 0.5598
Epoch 189/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0348 - acc: 0.5789
Epoch 190/200
418/418 [==============================] - 4s 10ms/step - loss: 0.0370 - acc: 0.5742
Epoch 191/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0361 - acc: 0.5622
Epoch 192/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0327 - acc: 0.5550
Epoch 193/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0346 - acc: 0.5837
Epoch 194/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0342 - acc: 0.5718
Epoch 195/200
418/418 [==============================] - 4s 11ms/step - loss: 0.0324 - acc: 0.5670
Epoch 196/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0328 - acc: 0.5646
Epoch 197/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0396 - acc: 0.5789
Epoch 198/200
418/418 [==============================] - 5s 11ms/step - loss: 0.0418 - acc: 0.5957
```

```
#@title Make predictions with the trained model

train_predict = RMS_model.predict(x_train)
test_predict = RMS_model.predict(x_test)

#test_predict = SGD_model.predict(x_test)
#test_predict = de_scale(test_predict, df)
#y_origin =de_scale(y_test, df)
```

Make predicti

```
#@title Plot Multi-step, Multi-feature predictions.
def predict_plot(df, y_predict, targets):
  """ Plot the multi-step, multi-result predictions.

  :param: df, pd.DataFrame, e.g. df = pd.read_excel("USMacroData.xls
  :param: y_predict, 2-dim np.array, the model-predicted values, in 
          In our example, look_forward = 3, number of target feature
  :param: targets, list, target features, e.g ["Inflation", "Unemplo
  """

  y_predict = de_vectorize(y_predict, 2, 3)
  assert y_predict.shape[1] == len(targets), "Incompatible size of t
  assert df.shape[0] == y_predict.shape[0], "Incompatible original d

  look_forward = y_predict.shape[2]

  for index, target in enumerate(targets):
    plt.figure(figsize=(17, 8))
```

Plot Multi-step

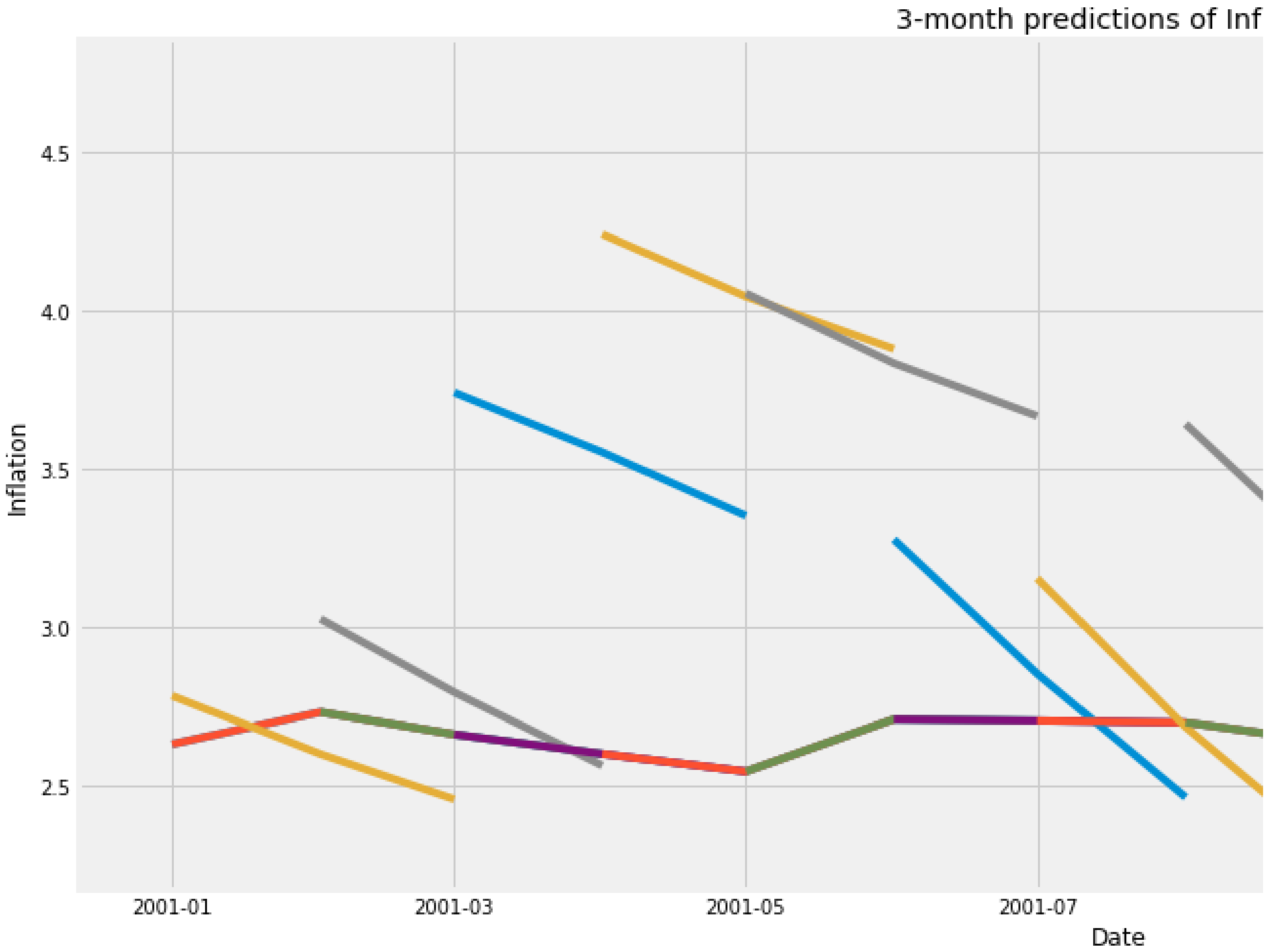
```
    plt.figure(figsize=(17, 8))
    plt.plot(df[0:12][target])
    for i in range(len(y_predict)):
      y = list(y_predict[i][index])
      x = list(df.index[i: i+look_forward])
      data = pd.DataFrame(list(zip(x, y)), columns =[df.index.name, 
      data = data.sort_values(df.index.name)
      data.set_index(df.index.name, inplace=True)

      if i < 12:
        plt.plot(df[i: i+look_forward][target])
        plt.plot(data)
        plt.xlabel("Date")
        plt.ylabel(target)
        plt.title("3-month predictions of " + target)
    plt.show()
```

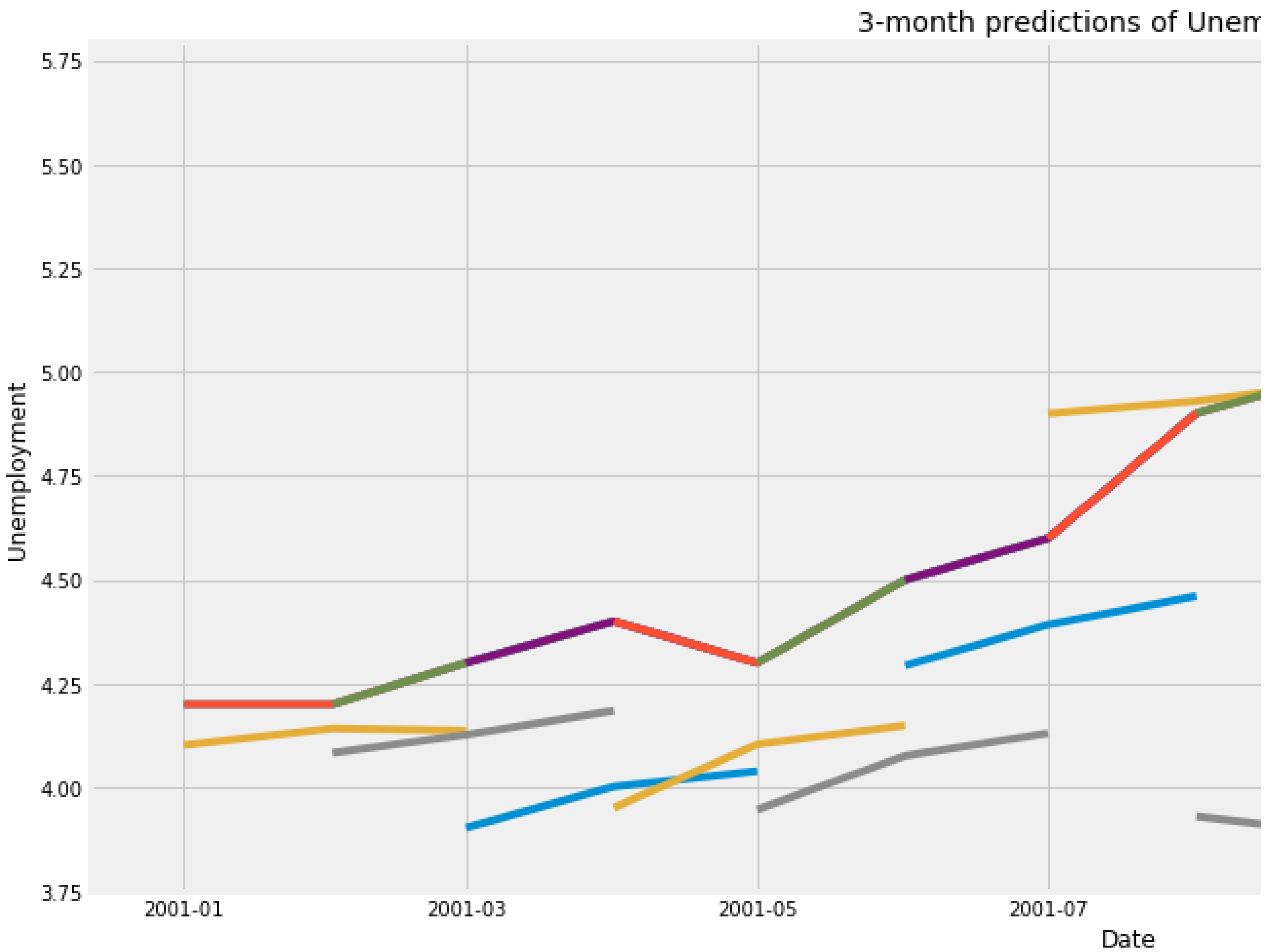
To read to graph below

- Each short line segment is a 3-month prediction: start, middle, end point of the line segment month's data respectively.
- X axies is the data.
- The long line is the real data.
- We plot the prediction for year 2001, change the parameter as you want to get prediction for

```
#@title We show the first 12 month's data and corresponding 3-month 
predict_plot(df[432:][["Inflation", "Unemployment"]], test_predict, 
```

We show the f month predict


3-month predictions of Inf
Inflation
4.5
4.0
3.5
3.0
2.5
2001-01
2001-03
2001-05
2001-07
Date


3-month predictions of Unem
Unemployment
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
2001-01
2001-03
2001-05
2001-07
Date

Data Analysis

Though the dataset is not big enough, we still successfully capture several features in the predicti

- **Model predictions shows similar trend as the real data**, e.g. from the prec predicted values are more or less in the most correct range and goes in the same direction a
- **The model captures the range of the real data very precisely**.
- **All 3-month predictions are continuous**, which means the modell successfully

Part II.3 Advanced model: Generative Adversaria

Generative Adversarial Networks (GAN) have been a successful model in genera **The idea that GANs can to used to predict time-series data is new and e** in learning characteristics of data, our model is based on the **assumptions**.

- Values of a **feature has certain patterns** and behavior (characteristics).
- **The future values of a feature should follow more or less the same pa** operating in a totally different way, or the economy drastically changes).

Our **goal** is that

- Generate future data that has similar (surely not exactly the same) distribution as the histori

In our model, we use

- **LSTM as a time-series generator**.
- **1-dimensional CNN as a discriminator**.

```
#@title imports

import keras
from keras.layers import Dense, Dropout, Input
from keras.models import Model,Sequential
from tqdm import tqdm
from keras.layers.advanced_activations import LeakyReLU
from keras.layers import LSTM, Conv1D, MaxPool1D, BatchNormalization
```

imports

```
#@title Data preparation

#@title Data preparation

def transform_data(df, features, targets, look_back = 0, look_forwar
  """transform the data in a custom form.

  :param: df, pd.DataFram, the data,
    e.g. df = pd.read_excel("USMacroData.xls", "All")
  :param: features, list of strs, the features to be uses as the sou
     e.g. ["Wage", "Consumption"]
  :param: look_back, int, number of days to look back in historic da
```

Data preparati

```
      e.g. look_back = 11 means we use the last (11+1)=12 months' data
    :param:look_forward, int, num of days to look forward
      e.g. look_forward = 3 means we want to predict next 3 months' da
    :param: split_ratio, float, split the data into training dataset a
      e.g. split_ratio=0.7 means we use the first 70% of the data as t
    :rtype: np.arrays, x_train, y_train, x_test, y_test
    """
    x, y = [], []
    for i in range(look_back, len(df) - look_forward):
        assert look_back < len(df)-look_forward, "Invalid look_back, l

        x.append(np.array(df[i-look_back : i+1][features]))
        y.append(np.array(df[i+1: i+look_forward+1][targets]).transpos

    # List to np.arrary
    x_arr = np.array(x)
    y_arr = np.array(y)

    split_point = int(len(x)*split_ratio)

    return x_arr[0:split_point], y_arr[0:split_point], x_arr[split_poi

features = ["Wage", "Consumption", "Investment", "InterestRate"]
targets = ["Inflation", "Unemployment"]
x_train, y_train, x_test, y_test = transform_data(df, features=featu

#Note that all returned np.arrays are three dimensional.
#Need to reshape y_train and y_test to fit the LSTM

# For the multi-step LSTM model only
y_train = np.reshape(y_train, (y_train.shape[0], -1))
y_test = np.reshape(y_test, (y_test.shape[0], -1))
```

```
#@title Make sure all data forms are as what we want

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

Make sure all

```
(418, 12, 4)
(418, 6)
(180, 12, 4)
(180, 6)
```

Model architecture: LSTM generator

It's a 1-layer LSTM model.

- 50 hidden layers of LSTM cells
- 1 dense layer with 6 (2*3) dimensional output, since we have 2 features and 3 months to pre

```
#@title Create generator
def create_generator():
```

Create genera

```
  generator = Sequential()
  generator.add(LSTM(50, input_shape=(12,4)))
  generator.add(Dense(6))
  generator.compile(loss="mean_squared_error", optimizer="RMSprop", 

  return generator

generator = create_generator()
generator.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 50)                11000
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 306
=================================================================
Total params: 11,306
Trainable params: 11,306
Non-trainable params: 0
_________________________________________________________________
```

Model architecture: CNN discriminator

The structure of the discriminator is given by

- Reshape layer. Each row in y_train is acturally 1-dimensional (6,), which is different from (6,1
- 1-dimensional Convolutional layer with 32, 3×1 filters to capture the characteristics of 3-mo
- LeakyReLU layer
- Dropout layer. Random reconfigurate 10% of the weights to zero to prevent overfitting.
- 1-dimensional Convolutional layer with 64, 3×1 filters to capture more characteristics of the
- Batchnormalization layer. To normalize the data.
- 1 Dense layer with 50 hidden nets.
- Dropout layer.
- 1 Dense layer with 1 net.

```
#@title Create discriminator
# CNN discriminator, Learn the distribution of the price.
# The goal of the gan model is to study the "characteristics" of, fo
# The generator tries to generate "Inflation" data as real as possib
FILTER_SIZE = 3
NUM_FILTER = 32
INPUT_SIZE = 3 # num of days we want to predict
MAXPOOL_SIZE = 1 # our data set is small, so we don't even need it
BATCH_SIZE = 1 # our data set is small, we don't need large batch si
STEP_PER_EPOCH = 612//BATCH_SIZE
EPOCHS = 10


def create_discriminator():
  discriminator = Sequential()
  discriminator.add(Reshape((6,1), input_shape=(6,)))
```

Create discrim

```
  discriminator.add(Conv1D(NUM_FILTER, FILTER_SIZE, input_shape = (6
  discriminator.add(LeakyReLU(0.2))
  discriminator.add(Dropout(0.1))
  discriminator.add(Conv1D(2*NUM_FILTER, FILTER_SIZE))
  discriminator.add(BatchNormalization())

  discriminator.add(Dense(units=50))
  discriminator.add(Dropout(0.1))

  #reduce the dimension of the model to 1
  discriminator.add(Flatten())

  discriminator.add(Dense(units=1, activation="sigmoid"))

  discriminator.compile(loss="mean_squared_error", optimizer="RMSpro
  return discriminator

discriminator = create_discriminator()
discriminator.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
Model: "sequential_7"
```

Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 6, 1)	0
conv1d_1 (Conv1D)	(None, 4, 32)	128
leaky_re_lu_1 (LeakyReLU)	(None, 4, 32)	0
dropout_1 (Dropout)	(None, 4, 32)	0
conv1d_2 (Conv1D)	(None, 2, 64)	6208
batch_normalization_1 (Batch	(None, 2, 64)	256
dense_6 (Dense)	(None, 2, 50)	3250
dropout_2 (Dropout)	(None, 2, 50)	0
flatten_1 (Flatten)	(None, 100)	0
dense_7 (Dense)	(None, 1)	101

```
Total params: 9,943
Trainable params: 9,815
Non-trainable params: 128
```

```
#@title Create a GAN model with LSTM as the generator and CNN as the
def create_gan(discriminator, generator):
    discriminator.trainable=False
```

Create a GAN
CNN as the di

```
    gan_input = Input(shape=(12,4))
    x = generator(gan_input)
    gan_output= discriminator(x)
    gan= Model(inputs=gan_input, outputs=gan_output)
    gan.compile(loss='mean_squared_error', optimizer='adam')
    return gan
gan = create_gan(discriminator, generator)
gan.summary()
```

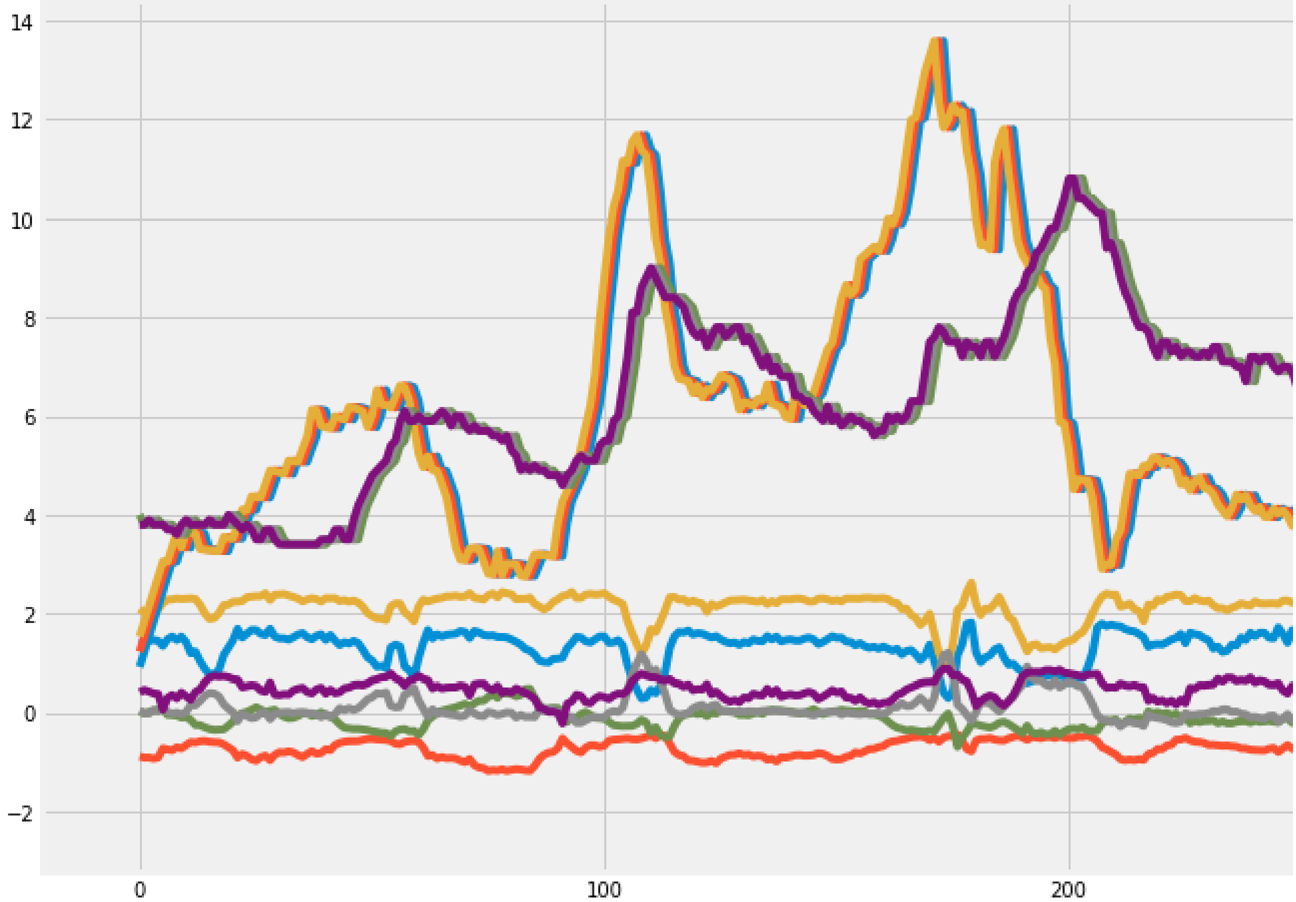
```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 12, 4)             0
_________________________________________________________________
sequential_6 (Sequential)    (None, 6)                 11306
_________________________________________________________________
sequential_7 (Sequential)    (None, 1)                 9943
=================================================================
Total params: 21,249
Trainable params: 11,306
Non-trainable params: 9,943
_________________________________________________________________
```

```
#@title Training function for the entangled GAN model
def training(x_train, y_train, x_test, y_test, epochs=1, random_size=

    #Loading the data
    random_count = 4*x_train.shape[0] / random_size

    # Creating GAN
    generator= create_generator()

    #y_lstm = np.reshape(y_train, (y_train.shape[0],1))
    #scores = generator.fit(x=x_train,y=y_lstm, batch_size=1, epochs
    #plt.plot(generator.predict(x_train))
    #plt.plot(y_lstm)
    #plt.show()


    discriminator= create_discriminator()
    gan = create_gan(discriminator, generator)

    for e in range(1,epochs+1 ):
        print("Epoch %d" %e)
        for index in tqdm(range(random_size)):
        #generate  random noise as an input  to  initialize the  gen
            feature = x_train[np.random.randint(low=0,high=x_train.sl

            # Generate fake MNIST images from noised input
            fake_money = generator.predict(feature)
            #print(fake_money)
            #print(fake_money.shape)
            # Get a random set of  real images

            #real_money =y_train[np.random.randint(low=0,high=y_trai
```

Training funct

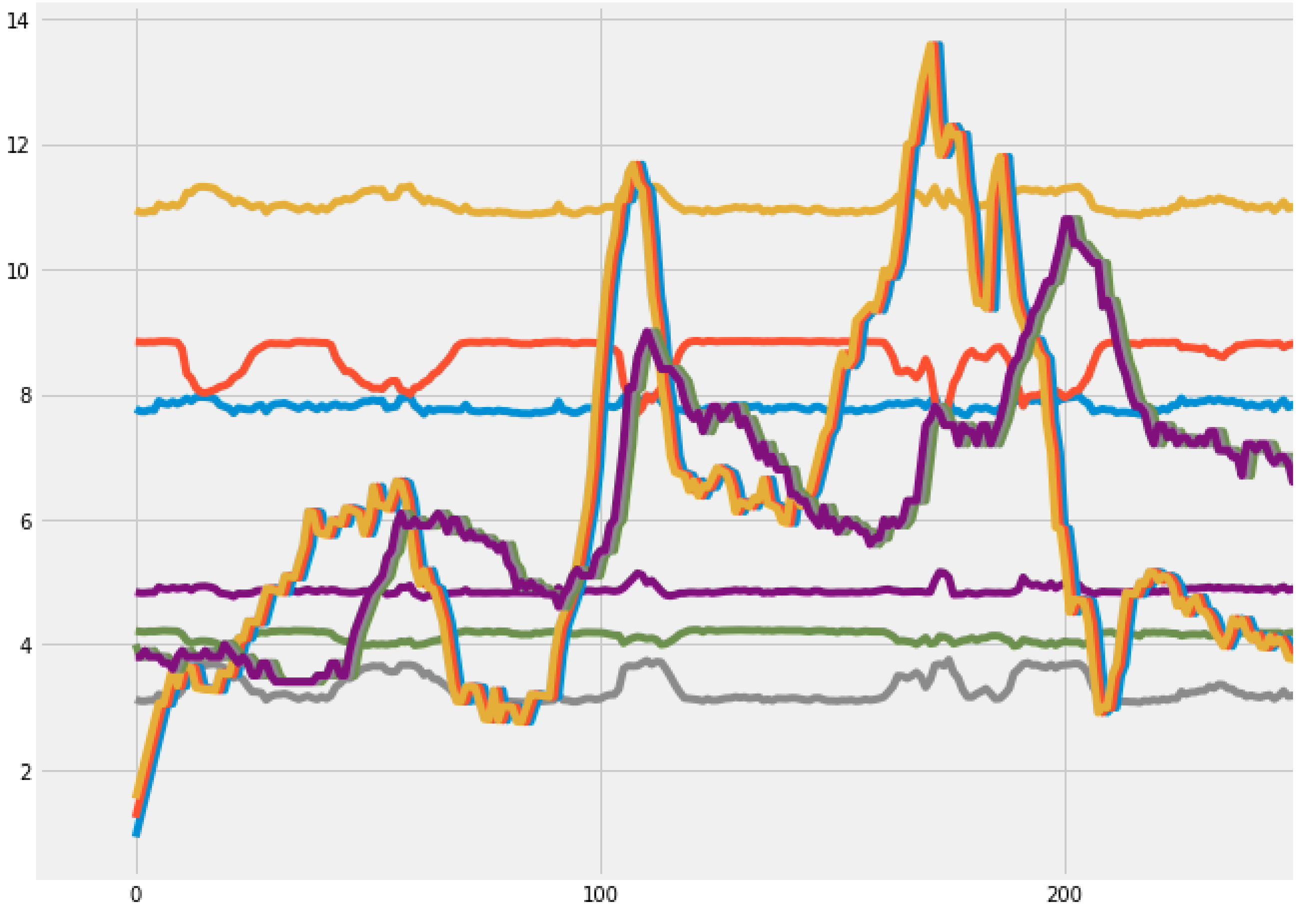
```
            upper_bound = int(np.random.randint(low=random_size, high

            real_money = y_train[upper_bound-random_size: upper_bound

            real_money = np.reshape(real_money, (real_money.shape[0]
            #print(real_money)
            #print(real_money.shape)

            #Construct different batches of  real and fake data
            combination = np.concatenate([real_money, fake_money])

            # Labels for generated and real data
            y_dis=np.zeros(2*random_size)
            y_dis[:random_size]=0.9

            #Pre train discriminator on  fake and real data  before s
            discriminator.trainable=True
            discriminator.train_on_batch(combination, y_dis)

            #Tricking the noised input of the Generator as real data
            trick_feature = x_train[np.random.randint(low=0,high=x_tr
            y_gen = np.ones(random_size)

            # During the training of gan,
            # the weights of discriminator should be fixed.
            #We can enforce that by setting the trainable flag
            discriminator.trainable=False

            #training  the GAN by alternating the training of the Dis
            #and training the chained GAN model with Discriminator's
            gan.train_on_batch(trick_feature, y_gen)

        if e == 1 or e % 20 == 0:
          #plot generator_predict(x_test) and y_test on the same grap
          plt.figure(figsize=(17,8))
          plt.plot(generator.predict(x_train))
          y_plot = np.reshape(y_train, (y_train.shape[0],6))
          plt.plot(y_plot)
          plt.show()

training(x_train, y_train, x_test, y_test, epochs=100, random_size=128)
```

```
  0%|          | 0/128 [00:00<?, ?it/s]Epoch 1
100%|██████████| 128/128 [00:08<00:00, 15.65it/s]
```



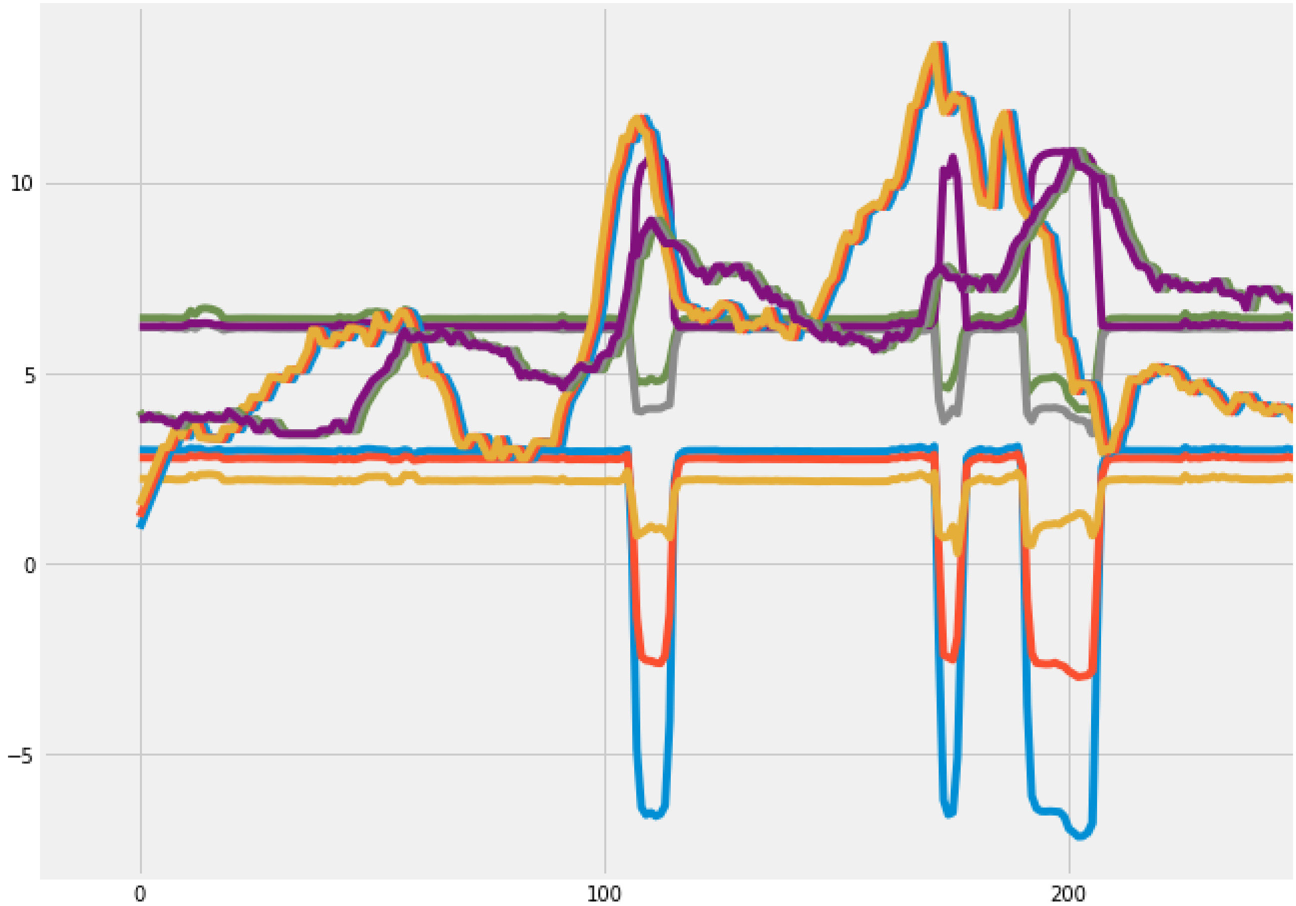

```
  2%|▏         | 2/128 [00:00<00:06, 19.94it/s]Epoch 2
100%|██████████| 128/128 [00:04<00:00, 29.42it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.94it/s]Epoch 3
100%|██████████| 128/128 [00:04<00:00, 29.58it/s]
  2%|▏         | 3/128 [00:00<00:04, 27.07it/s]Epoch 4
100%|██████████| 128/128 [00:04<00:00, 29.35it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.72it/s]Epoch 5
100%|██████████| 128/128 [00:04<00:00, 29.83it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.50it/s]Epoch 6
100%|██████████| 128/128 [00:04<00:00, 30.25it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.87it/s]Epoch 7
100%|██████████| 128/128 [00:04<00:00, 30.41it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.67it/s]Epoch 8
100%|██████████| 128/128 [00:04<00:00, 30.37it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.97it/s]Epoch 9
100%|██████████| 128/128 [00:04<00:00, 30.58it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.65it/s]Epoch 10
100%|██████████| 128/128 [00:04<00:00, 30.07it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.40it/s]Epoch 11
100%|██████████| 128/128 [00:04<00:00, 30.25it/s]
  2%|▏         | 3/128 [00:00<00:04, 26.66it/s]Epoch 12
100%|██████████| 128/128 [00:04<00:00, 30.41it/s]
  3%|▎         | 4/128 [00:00<00:04, 29.75it/s]Epoch 13
100%|██████████| 128/128 [00:04<00:00, 29.22it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.95it/s]Epoch 14
100%|██████████| 128/128 [00:04<00:00, 30.80it/s]
  3%|▎         | 4/128 [00:00<00:03, 33.43it/s]Epoch 15
100%|██████████| 128/128 [00:04<00:00, 31.35it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.34it/s]Epoch 16
100%|██████████| 128/128 [00:04<00:00, 30.07it/s]
  3%|▎         | 4/128 [00:00<00:03, 33.40it/s]Epoch 17
100%|██████████| 128/128 [00:04<00:00, 31.02it/s]
```

```
  3%|▎         | 4/128 [00:00<00:04, 30.01it/s]Epoch 18
100%|██████████| 128/128 [00:04<00:00, 30.52it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.46it/s]Epoch 19
100%|██████████| 128/128 [00:04<00:00, 31.62it/s]
  3%|▎         | 4/128 [00:00<00:03, 34.89it/s]Epoch 20
100%|██████████| 128/128 [00:04<00:00, 31.69it/s]
```



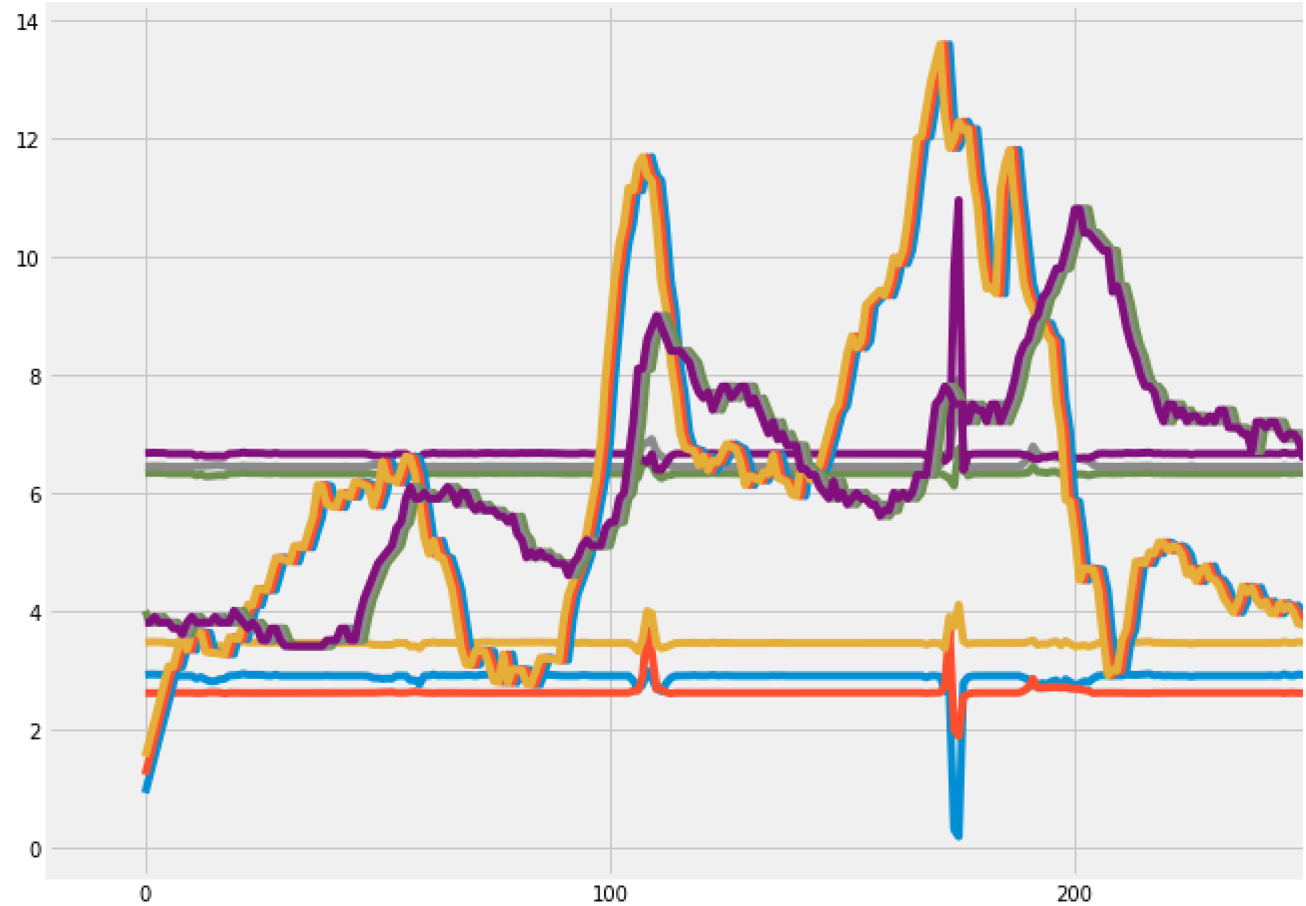

```
  2%|▏         | 2/128 [00:00<00:06, 19.48it/s]Epoch 21
100%|██████████| 128/128 [00:04<00:00, 29.86it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.29it/s]Epoch 22
100%|██████████| 128/128 [00:04<00:00, 29.05it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.60it/s]Epoch 23
100%|██████████| 128/128 [00:04<00:00, 29.78it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.10it/s]Epoch 24
100%|██████████| 128/128 [00:04<00:00, 29.62it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.07it/s]Epoch 25
100%|██████████| 128/128 [00:04<00:00, 29.27it/s]
  2%|▏         | 3/128 [00:00<00:04, 27.76it/s]Epoch 26
100%|██████████| 128/128 [00:04<00:00, 29.74it/s]
  2%|▏         | 3/128 [00:00<00:04, 26.23it/s]Epoch 27
100%|██████████| 128/128 [00:04<00:00, 28.71it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.21it/s]Epoch 28
100%|██████████| 128/128 [00:04<00:00, 29.99it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.93it/s]Epoch 29
100%|██████████| 128/128 [00:04<00:00, 30.66it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.86it/s]Epoch 30
100%|██████████| 128/128 [00:04<00:00, 29.74it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.26it/s]Epoch 31
100%|██████████| 128/128 [00:04<00:00, 30.43it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.09it/s]Epoch 32
100%|██████████| 128/128 [00:04<00:00, 30.46it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.95it/s]Epoch 33
100%|██████████| 128/128 [00:04<00:00, 29.99it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.55it/s]Epoch 34
100%|██████████| 128/128 [00:04<00:00, 29.70it/s]
```

```
  2%|▏         | 3/128 [00:00<00:04, 28.76it/s]Epoch 35
100%|██████████| 128/128 [00:04<00:00, 29.75it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.67it/s]Epoch 36
100%|██████████| 128/128 [00:04<00:00, 29.09it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.19it/s]Epoch 37
100%|██████████| 128/128 [00:04<00:00, 29.67it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.92it/s]Epoch 38
100%|██████████| 128/128 [00:04<00:00, 28.88it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.82it/s]Epoch 39
100%|██████████| 128/128 [00:04<00:00, 29.19it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.61it/s]Epoch 40
100%|██████████| 128/128 [00:04<00:00, 29.19it/s]
```



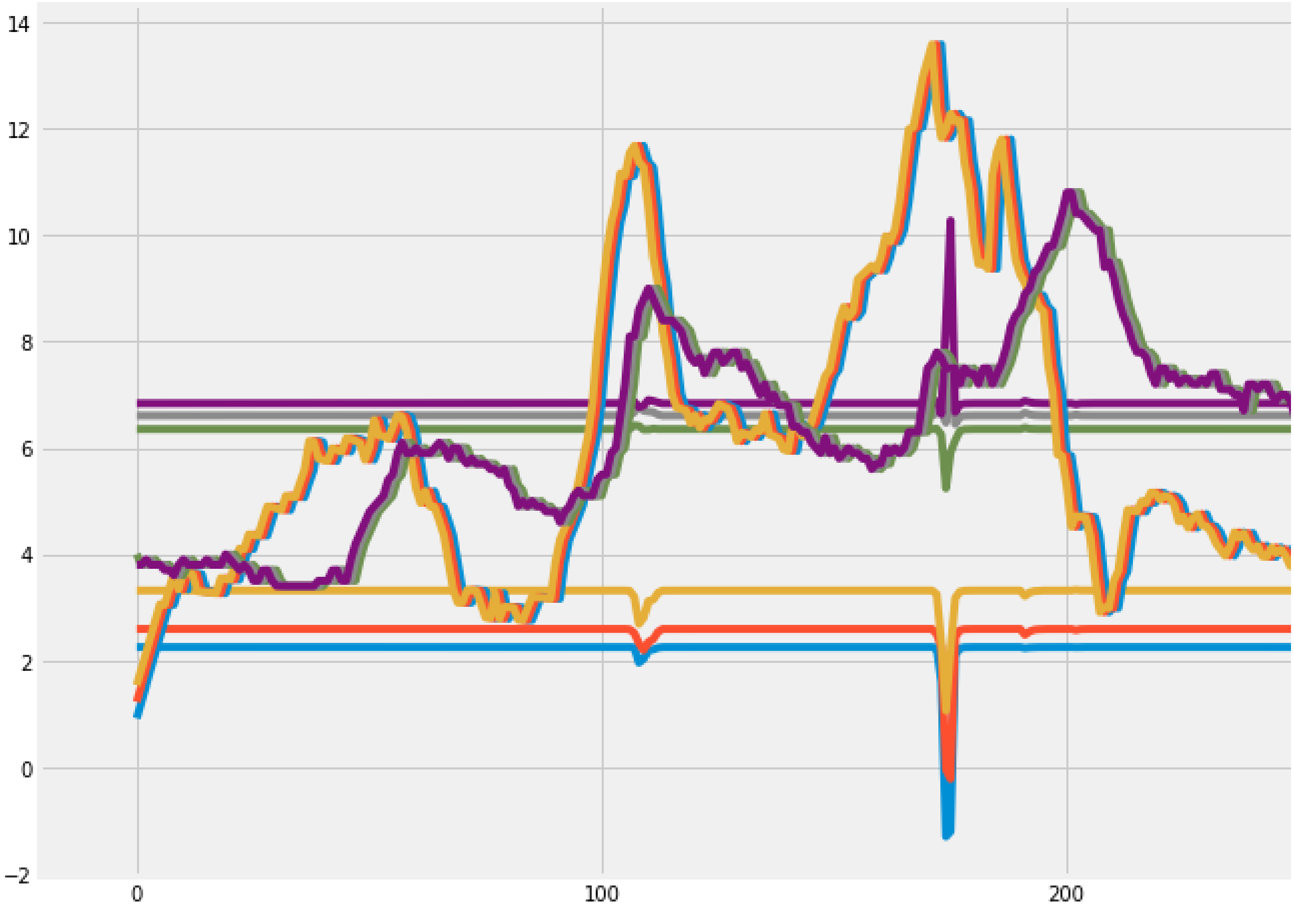

```
  2%|▏         | 2/128 [00:00<00:06, 19.26it/s]Epoch 41
100%|██████████| 128/128 [00:04<00:00, 29.63it/s]
  3%|▎         | 4/128 [00:00<00:03, 33.59it/s]Epoch 42
100%|██████████| 128/128 [00:04<00:00, 30.28it/s]
  3%|▎         | 4/128 [00:00<00:03, 32.75it/s]Epoch 43
100%|██████████| 128/128 [00:04<00:00, 30.83it/s]
  3%|▎         | 4/128 [00:00<00:03, 32.29it/s]Epoch 44
100%|██████████| 128/128 [00:04<00:00, 32.71it/s]
  3%|▎         | 4/128 [00:00<00:03, 32.79it/s]Epoch 45
100%|██████████| 128/128 [00:04<00:00, 28.47it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.92it/s]Epoch 46
100%|██████████| 128/128 [00:04<00:00, 30.41it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.85it/s]Epoch 47
100%|██████████| 128/128 [00:04<00:00, 29.80it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.54it/s]Epoch 48
100%|██████████| 128/128 [00:04<00:00, 30.91it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.64it/s]Epoch 49
100%|██████████| 128/128 [00:04<00:00, 29.83it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.82it/s]Epoch 50
100%|██████████| 128/128 [00:04<00:00, 29.02it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.06it/s]Epoch 51
100%|██████████| 128/128 [00:04<00:00, 28.55it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.80it/s]Epoch 52
```

```
  2%|▏         | 3/128 [00:00<00:04, 28.80it/s]Epoch 52
100%|██████████| 128/128 [00:04<00:00, 29.09it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.35it/s]Epoch 53
100%|██████████| 128/128 [00:04<00:00, 29.70it/s]
  2%|▏         | 3/128 [00:00<00:04, 27.59it/s]Epoch 54
100%|██████████| 128/128 [00:04<00:00, 30.46it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.65it/s]Epoch 55
100%|██████████| 128/128 [00:04<00:00, 30.19it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.88it/s]Epoch 56
100%|██████████| 128/128 [00:04<00:00, 29.89it/s]
  3%|▎         | 4/128 [00:00<00:03, 33.61it/s]Epoch 57
100%|██████████| 128/128 [00:04<00:00, 30.19it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.45it/s]Epoch 58
100%|██████████| 128/128 [00:04<00:00, 30.63it/s]
  2%|▏         | 3/128 [00:00<00:04, 27.55it/s]Epoch 59
100%|██████████| 128/128 [00:04<00:00, 30.14it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.68it/s]Epoch 60
100%|██████████| 128/128 [00:04<00:00, 29.85it/s]
```



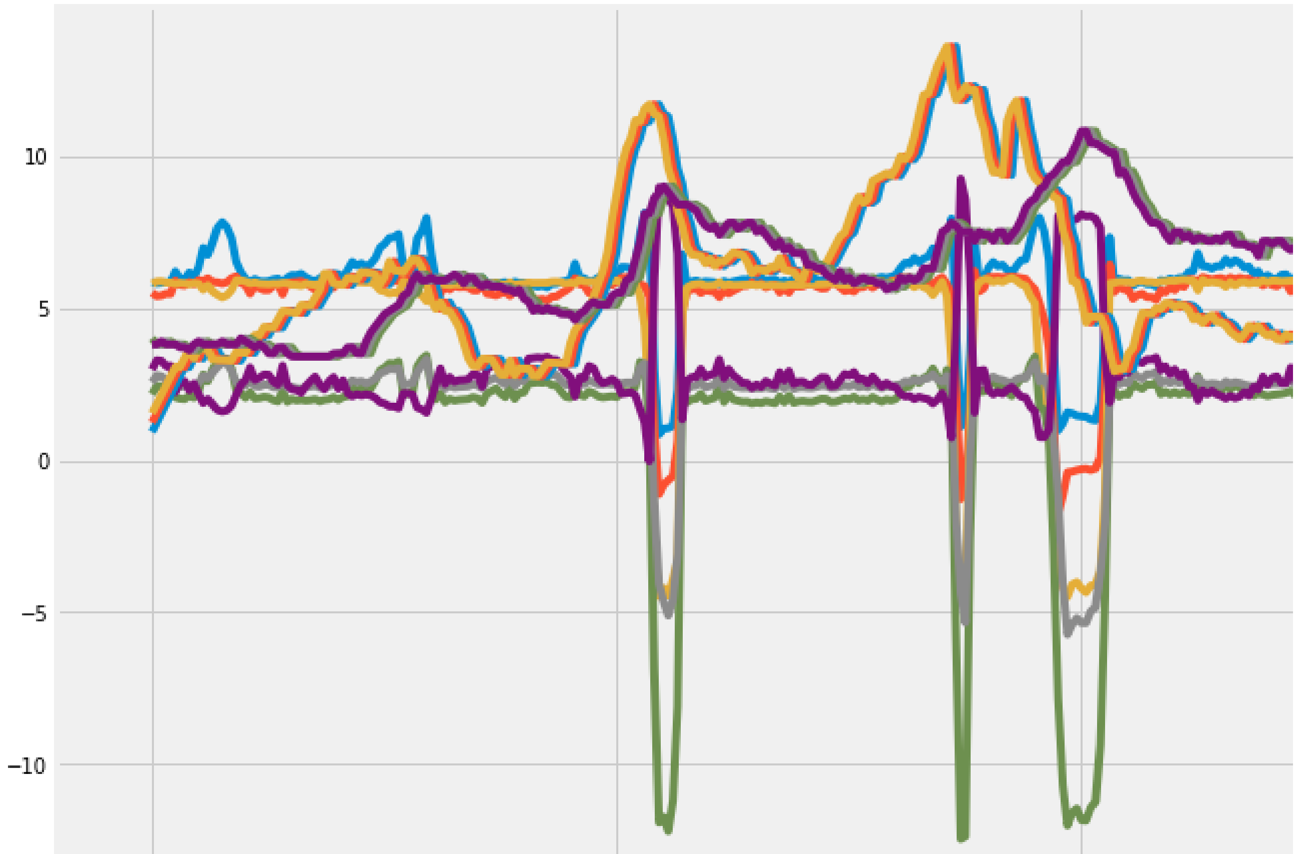

```
  2%|▏         | 2/128 [00:00<00:06, 19.16it/s]Epoch 61
100%|██████████| 128/128 [00:04<00:00, 30.38it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.39it/s]Epoch 62
100%|██████████| 128/128 [00:04<00:00, 29.60it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.60it/s]Epoch 63
100%|██████████| 128/128 [00:04<00:00, 29.58it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.27it/s]Epoch 64
100%|██████████| 128/128 [00:04<00:00, 30.06it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.11it/s]Epoch 65
100%|██████████| 128/128 [00:04<00:00, 30.36it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.81it/s]Epoch 66
100%|██████████| 128/128 [00:04<00:00, 30.15it/s]
  3%|▎         | 4/128 [00:00<00:03, 32.00it/s]Epoch 67
100%|██████████| 128/128 [00:04<00:00, 30.40it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.03it/s]Epoch 68
100%|██████████| 128/128 [00:04<00:00, 29.16it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.40it/s]Epoch 69
```

```
100%|██████████| 128/128 [00:04<00:00, 28.71it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.61it/s]Epoch 70
100%|██████████| 128/128 [00:04<00:00, 30.25it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.97it/s]Epoch 71
100%|██████████| 128/128 [00:04<00:00, 29.77it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.25it/s]Epoch 72
100%|██████████| 128/128 [00:04<00:00, 29.83it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.63it/s]Epoch 73
100%|██████████| 128/128 [00:04<00:00, 29.61it/s]
  3%|▎         | 4/128 [00:00<00:03, 33.55it/s]Epoch 74
100%|██████████| 128/128 [00:04<00:00, 29.75it/s]
  3%|▎         | 4/128 [00:00<00:03, 33.34it/s]Epoch 75
100%|██████████| 128/128 [00:04<00:00, 28.44it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.63it/s]Epoch 76
100%|██████████| 128/128 [00:04<00:00, 29.48it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.93it/s]Epoch 77
100%|██████████| 128/128 [00:04<00:00, 29.92it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.83it/s]Epoch 78
100%|██████████| 128/128 [00:04<00:00, 29.52it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.81it/s]Epoch 79
100%|██████████| 128/128 [00:04<00:00, 29.02it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.71it/s]Epoch 80
100%|██████████| 128/128 [00:04<00:00, 28.87it/s]
```




```
  2%|▏         | 3/128 [00:00<00:05, 24.81it/s]Epoch 81
100%|██████████| 128/128 [00:04<00:00, 29.22it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.41it/s]Epoch 82
100%|██████████| 128/128 [00:04<00:00, 29.89it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.14it/s]Epoch 83
100%|██████████| 128/128 [00:04<00:00, 29.72it/s]
  2%|▏         | 3/128 [00:00<00:04, 27.89it/s]Epoch 84
100%|██████████| 128/128 [00:04<00:00, 30.30it/s]
  3%|▎         | 4/128 [00:00<00:03, 33.79it/s]Epoch 85
100%|██████████| 128/128 [00:04<00:00, 29.43it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.43it/s]Epoch 86
```

```
100%|██████████| 128/128 [00:04<00:00, 30.24it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.16it/s]Epoch 87
100%|██████████| 128/128 [00:04<00:00, 29.39it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.31it/s]Epoch 88
100%|██████████| 128/128 [00:04<00:00, 30.50it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.11it/s]Epoch 89
100%|██████████| 128/128 [00:04<00:00, 31.16it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.89it/s]Epoch 90
100%|██████████| 128/128 [00:04<00:00, 30.44it/s]
  3%|▎         | 4/128 [00:00<00:04, 29.90it/s]Epoch 91
100%|██████████| 128/128 [00:04<00:00, 30.20it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.57it/s]Epoch 92
100%|██████████| 128/128 [00:04<00:00, 29.56it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.85it/s]Epoch 93
100%|██████████| 128/128 [00:04<00:00, 30.54it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.71it/s]Epoch 94
100%|██████████| 128/128 [00:04<00:00, 29.20it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.58it/s]Epoch 95
100%|██████████| 128/128 [00:04<00:00, 29.72it/s]
  2%|▏         | 3/128 [00:00<00:04, 27.01it/s]Epoch 96
100%|██████████| 128/128 [00:04<00:00, 29.18it/s]
  3%|▎         | 4/128 [00:00<00:04, 30.88it/s]Epoch 97
100%|██████████| 128/128 [00:04<00:00, 30.39it/s]
  3%|▎         | 4/128 [00:00<00:03, 31.04it/s]Epoch 98
100%|██████████| 128/128 [00:04<00:00, 30.20it/s]
  2%|▏         | 3/128 [00:00<00:04, 29.26it/s]Epoch 99
100%|██████████| 128/128 [00:04<00:00, 30.30it/s]
  2%|▏         | 3/128 [00:00<00:04, 28.76it/s]Epoch 100
100%|██████████| 128/128 [00:04<00:00, 30.81it/s]
```



Data Analysis

- **Our LSTM generator is not pre-trained**, which means **The GAN model** get results as good as the previous models, but **this experimental model shows p**

- The GAN model successfully learned the correct range.
- **The GAN model learns the most drastic characteristics of the data**. I

Part III Conclusions and Next steps

Conclusions

In this project on analyzing and forecasting the US macro data we managed to accomplish the fol

- **Statistical analysis in Part I**
  - Basic manipulation: read the file, find null values and set index and some column plott
  - Correlation analysis: compute different correlations and use to to validate our choice c
  - Time series analysis with `ARIMA`: grid search for optimal parameters and train the ARI
- **Build 3 Deep learning models from basic one to advanced one in Par**
  - Basic model: single-step, single-feature forecasting with LSTM
  - Generalized model: multi-step, multi-feature forcasting with LSTM
  - Advanced model: Generative Adversarial Network (GAN) with LSTM and CNN.

Along the way, we find several remarkable patterns and features of our data

- Features show **long-period seasonality**.
- **Several features show apparent correlations**.
- **Most features slightly leads the Inflation feature**.
- The GAN model successfully learned the correct range.
- **The GAN model learns the most drastic characteristics of the data**.

Next steps

The USMacroData is not a big dataset, the following are a few furthur steps that we have done bu directions we can try to investigate:

- It's natural to **include the target feature itself into consideration**, because i most relevant to the future value of the feature itself. It can be easily achieved by modifying
- Investigate the difference between the first thirty years and the last twenty years.
- **Pre-train the LSTM model in the GAN model. In this way, the mod**
  .